DESIGN AND IMPLEMENTATION OF A RING INTERFACE/HOST ADAPTER FOR AN IBM SYSTEM 360

Eberhard Otto Wortmann

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

DESIGN AND IMPLEMENTATION OF A
RING INTERFACE/HOST ADAPTER
FOR AN IBM SYSTEM 360

by

Eberhard Otto Wortmann

June 1974

Thesis Advisor: R.H. Brubaker,Jr.

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Design and Implementation of a Ring Interface/Host Adapter for an IBM System 360		5. TYPE OF REPORT & PERIOD COVERED Master's Thesis; June 1974
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Eberhard Otto Wortmann		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		12. REPORT DATE June 1974
		13. NUMBER OF PAGES 86
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Naval Postgraduate School Monterey, California 93940		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Ring communication network
ring interface/host adapter

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

At the Naval Postgraduate School a project is underway to develop a ring communication network which will eventually connect various computer facilities on the campus. The main emphasis is put on modularity to increase design flexibility and keep cost low. Therefore all host/ring interface functions are performed by a general purpose Ring Interface which then is adapted to its specific host by a device called

(20. ABSTRACT continued)

the "Ring Interface/Host Adapter." Here the design and implementation of an adapter is described that matches the Ring Interface to the Naval Postgraduate School's IBM System 360/67. The heart of the adapter is a programmed control unit or "microcontroller" with an assembler-level programming language, SMAL.

Design and Implementation of a
Ring Interface/Host Adapter
for an IBM System 360

by

Eberhard Otto Wortmann
Lieutenant Commander, Federal German Navy
Ing.(grad.) Fachhochschule Hamburg, 1971

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1974

ABSTRACT

At the Naval Postgraduate School a project is underway to develop a ring communication network which will eventually connect various computer facilities on the campus. The main emphasis is put on modularity to increase design flexibility and keep cost low. Therefore all host/ring interface functions are performed by a general purpose Ring Interface which then is adapted to its specific host by a device called the "Ring Interface/Host Adapter." Here the design and implementation of an adapter is described that matches the Ring Interface to the Naval Postgraduate School's IBM System 360/67. The heart of the adapter is a programmed control unit or "microcontroller" with an assembler-level programming language, SMAL.

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

A. THE BASIC CONCEPT

1. Initial Considerations

In recent years the ideal of "modularization" has gained much popularity in the area of Computer Science because of its advantages with respect to design flexibility and reduction of cost. Since cost and flexibility were main considerations in this project, heavy emphasis was placed on a modular approach. In addition to this, software (or "firmware") was to replace hardware wherever possible since it could be produced locally at low cost and it would further increase design flexibility.

2. Basic Design Decisions

Figure 1 shows a conceivable ring communication configuration, where a "node" is defined as a host processor together with its ring interface. Though different processors would be connected to the ring, the functions performed by each RI were to be the same at all nodes:

- Data and control tokens traveling along the ring had to be received, evaluated, and retransmitted.
- Certain checking functions had to be performed and status information had to be sent to the host processor.
- Control signals from the host processor had to be acknowledged and complied with.

Therefore, the concept of a Ring Interface eventually evolved, which would incorporate all these functions in the most

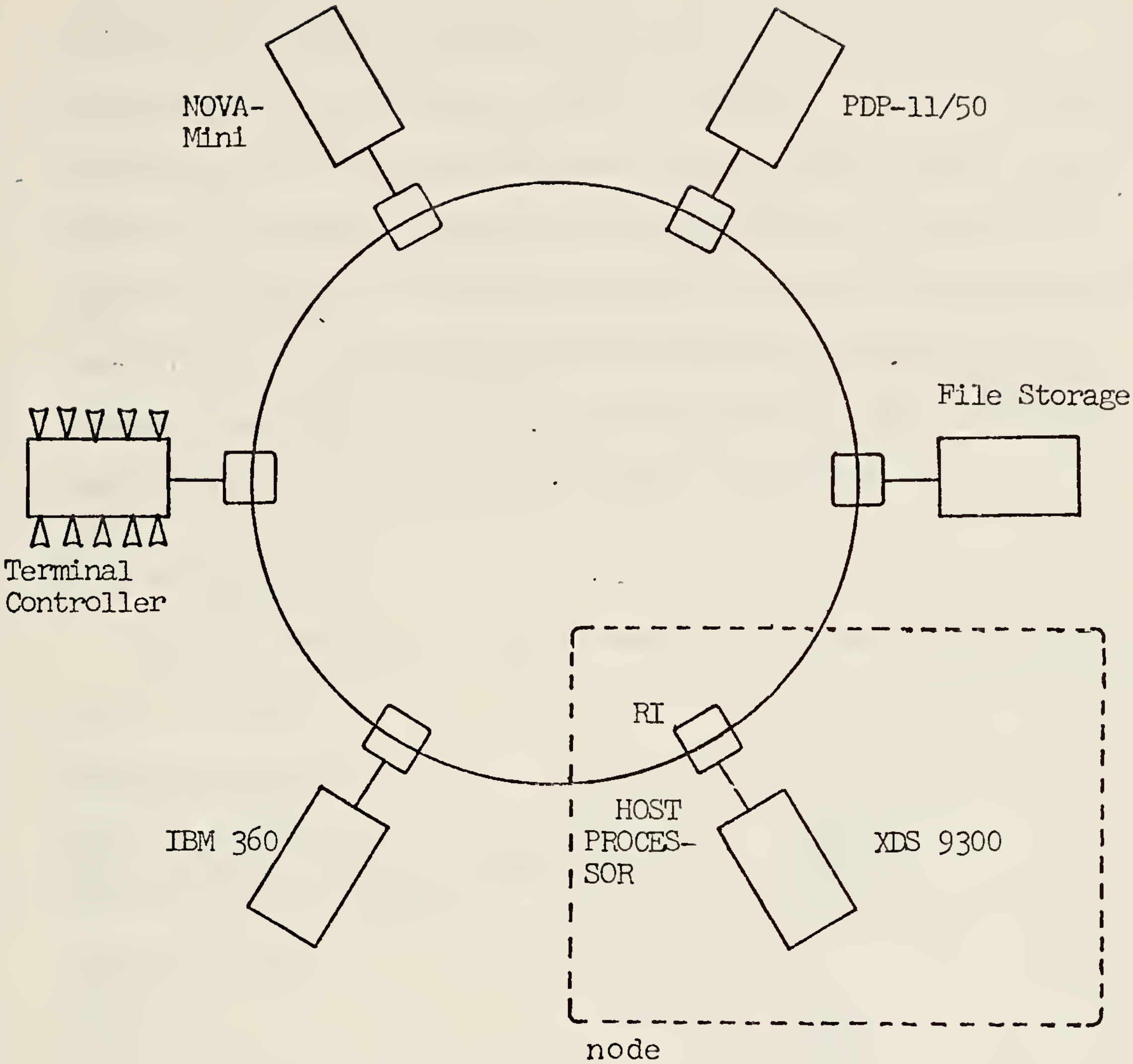


Fig. 1. Envisioned Ring Communication Network

efficient manner independent of any host processor. In consequence of this each host processor would be communicating with its Ring Interface via a device which would adapt the general purpose Ring Interface to the host's specific needs. (Some hosts may be directly connectable to the RI, and programmatically execute the necessary control sequences.) The unit performing this role will be called Ring Interface/Host Adapter (RIHA).

B. TERMINOLOGY

Where adequate in this text the following abbreviations will be used:

<u>Hardware Units</u>

Ring Interface	RI
Ring Interface/Host Adapter	RIHA
Parallel Data Adapter	PDA

<u>Control Lines</u>

Receive	RCV
Ring Data Ready	RDR
Host Accept	HA
Transmit	XMIT
Ring Demand	RID
Host Data Ready	HDR
Alter Process Name	APN
Write Name	WRTN
Disconnect	DISC
RI Reset	RESET
Receive CRC Error	RCRC
Receive Overrun	ROVR
Transmit CRC Error	XCRC
Transmit Overrun	XOVR
Message Bit 1	MSB1
Message Bit 2	MSB2
Ring Error	RERR
Ring Disconnected	RDISC
Ring Data Out (8)	RDO
Ring Data In (8)	RDI

To facilitate understanding the following terms will be used in a unique sense throughout this text.

<u>TRANSMIT-SEQUENCE</u>:	Transfer of data from PDA to RI
ACCEPT:	Transfer of data from PDA into RIHA
DELIVER:	Transfer of data from RIHA to RI
<u>RECEIVE-SEQUENCE</u>:	Transfer of data from RI to PDA
RECEIVE:	Transfer od data from RI into RIHA
RELEASE:	Transfer of data from RIHA to PDA

II. DEFINITION OF THE RING INTERFACE/HOST ADAPTER

Figure 2 shows the conceptual configuration of a RIHA consisting of a Ring Interface (RI) attached to the ring and an I/O performing part of the host processor on the other end with the adapter in between in the role of an interpreter.

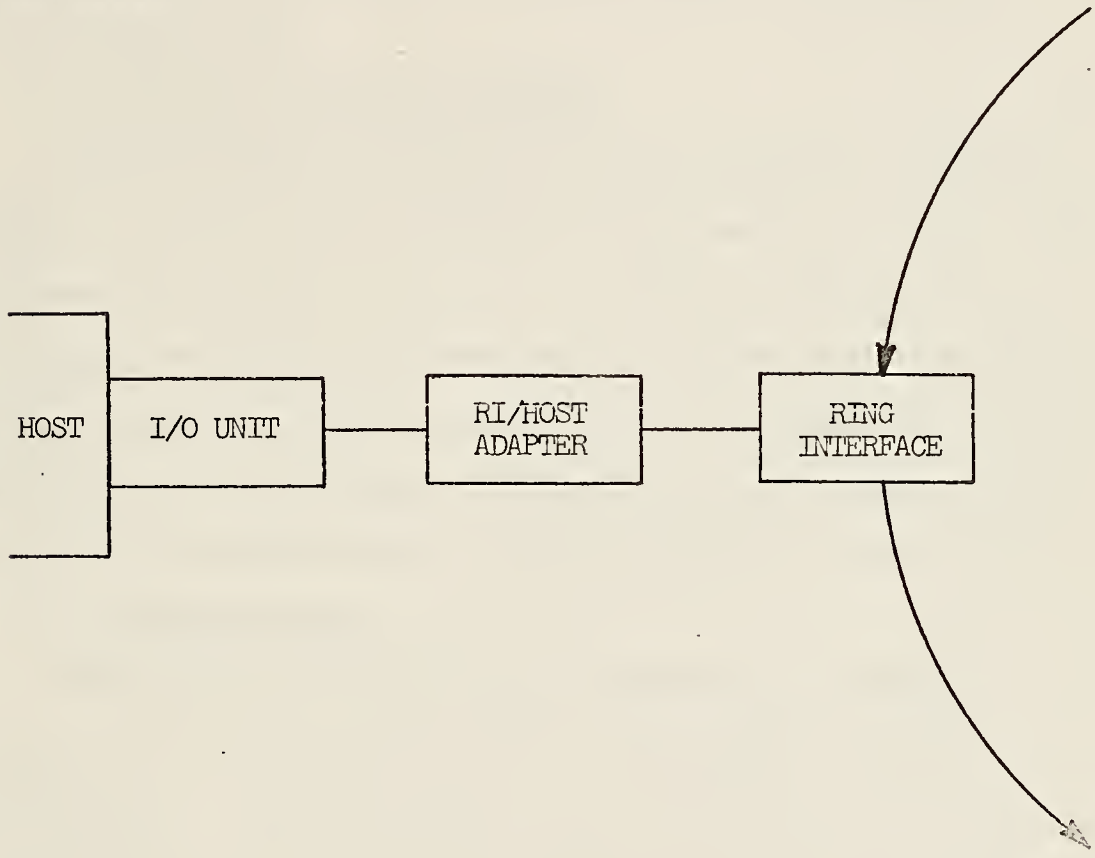


Fig. 2. Conceptual Configuration of a Ring Interface/Host Adapter

As mentioned in the introduction all functional requirements to connect any host to the ring will be performed by the Ring Interface. While exploring the necessary exchange of information between Ring Interface and Host on a conceptual

level, the range of tasks the Adapter has to handle will be defined.

A. HOST PROCESSOR CONTROL OF RING INTERFACE

To enable the host processor and the Ring Interface (RI) to communicate successfully with each other and to execute required procedures, certain control sequences must be established.

1. Connect/Disconnect Sequence

This sequence provides the host processor with the ability to inform its RI that for some reason the host wants to disconnect from the ring. The required action on the part of the RI will be to step out of the ring by providing a route to the ring data by-passing this interface. At any later time, a signal sequence issued by some process inside the host can cause the RI to switch itself into the ring.

2. Reset Sequence

When the host ties up the ring with too long a message or by an error, the RI will disconnect from the ring automatically. The only way to get it back into the ring is by notifying the RIHA. (For more details see discussion of RI-Control Lines.)

3. Alter Process Name

One of the RI tasks is to constantly watch whether a message being transmitted onto the ring by any other RI is addressed to a process residing at its host. For this

purpose the RI keeps a list of process names. One signal sequence the host must be able to send to the RI will therefore contain the name of a process and the command either to place this name into its associative memory of process names or to delete it from the list.

Before switching the RI into the ring the normal procedure would be to delete all possible process names and set the list to the new valid names. It is essential that all names be deleted after a power-on sequence, since the memory contents are random at that time.

B. DATA TRANSFER

While data and control tokens on the ring move in one direction only, information between the RI and the host will go both ways. The Adapter therefore will have to handle three situations:

1. The Receive Sequence

When the RI detects a message on the ring whose address header specifies a process residing at its host, it alerts the host to receive it: the Adapter activates a Receive Sequence.

2. The Transmit Sequence

When the host intends to transmit a message to a process residing at one or several of the other nodes it signals the RI about it: the Adapter activates a Transmit Sequence.

3. Interference of Receive and Transmit

When either the RI wants the host to receive a message from the ring while the host is waiting to get a message transmitted or when the RI has already asked for a Receive-Sequence when the host wants to initiate a Transmit Sequence: the Adapter has to be equipped to make a decision which to handle first.

III. THE PLANNING PHASE

A. PRELIMINARY CONSIDERATIONS

Before the author started design work on this Adapter, a thesis on a prototype ring-structured computer network had been completed by Hirt [3]. In their research for ways to systematize the overall approach, members of the Computer Science Group at this school developed the idea that for the design of a standardized RI as well as for building the adapters for the different computers employed by various academic departments, the availability of a general purpose microcontroller, programmable to diverse applications would simplify the design as well as the testing of these devices and would accelerate the whole project. Therefore such a controller was developed by Brubaker with Harris [1].

As further steps in an organized approach a language called SMAL evolved to facilitate programming each microcontroller and an assembler for this language was written by Kildall [2] in PL/M [4] to run on the Intellec-8 or Intellec 80 developmental system [5].

B. ORIGINAL APPROACH

Since thesis work on the standardized ring interface [6] and this Adapter was begun at the same time, the exact requirements of the RI were not initially available.

Therefore, emphasis of this thesis was first placed on investigating the host's I/O requirements, in this case the multiplexor channel of an IBM System 360/67. An IBM OEM interface manual [7] was used as a source of information. It was decided to build the Adapter in such a way that it would connect to the IBM channel as an IBM Control Unit. Since it would not be possible to test the Adapter by connecting it to the channel in its system environment because of IBM hardware regulations and user demand on the System 360, a program was written in PL/M for an MCS-8 microcomputer which incorporated the channel logic and would serve to test the Adapter by simulating the channel.

After a number of weeks on this approach, the Naval Postgraduate School's Computer Facility received word that it would be able to acquire an IBM 2701 Transmission Control Unit with a Parallel Data Adapter [8,9]. This would

1. reduce the complexity of the RI/Host Adapter
2. simplify the electrical requirements and standards.

C. REVISED APPROACH

Under these circumstances a new start was made. The host's requirements were taken from IBM manuals about the Parallel Data Adapter. The Ring Interface's control and data lines were defined by now and the paper about the microcontroller was available [1].

IV. REALIZATION OF A RI/HOST ADAPTER

To gain some first hand experience in this area the author assembled one of the microcontrollers on a bread-board using a wire wrapping technique. After this first encounter with integrated logic chips the design of the actual RIHA began.

A. GENERAL OUTLINE

1. The Interface and Logic Support

The control and data lines between the RIHA and its environment were predefined by the requirements of PDA and RI. On the inside there would be the microcontroller, treated here as a black box, handling all sequencing re-quirements through the ability to test the logic state of incoming lines, to handle the sequencing of actions according to these test results and its program, and to strobe certain outgoing lines as required by the program while supplying relevant data on its 8-bit data out bus.

2. Speed Considerations - The FIFO Buffer

One area where differences between the RI and its host become apparent is their different speed. The RI has to watch traffic on the ring either until a message for its host arrives or until it obtains the ring to transmit a message of its host onto the ring. When it eventually starts to receive or transmit, its speed is determined completely by the speed maintained on the ring. A byte of

data assembled from the ring and ready to be transferred toward the host which is not accepted before the next byte is ready for transfer constitutes an overrun condition. Also, the last bit of a byte transmitted into the ring with the next byte not yet available from the RI/Host Adapter will cause an overrun error. Either case will necessitate a retransmission of the message involved. On the Host side of the RI/Host Adapter, acceptance or release of data depends on the availability of the channel, which again is affected by requests of other I/O devices supported by the same channel. While the channel (and with it the PDA) is normally faster than the Ring and is capable of asynchronous, byte-by-byte conversation, it might be absent for an amount of time causing an overrun error at the RI.

Not to do anything about this problem and leave it open to chance was considered an unrealistic solution, since frequent retransmission of a message would degrade performance of the whole system. One way to solve the problem would have been the adapter to include a buffer memory into which an incoming message would be written and only after the complete message had been recorded it would be sent out the other end. This way complete independence of RI and PDA would be attaned. On the other hand, message length on the ring would be limited by the size of the buffer in the adapter. The third way, and the one finally chosen, consists of a first-in-first-out (FIFO) buffer memory of

size 1024 x 8 bits. Since many messages on the ring are expected to be shorter than 1024 bytes, for a large part of the data transfer the advantage of independence of the ring from the speed of the channel is conserved without limiting transfer of data files or long messages. The FIFO serves to smooth out the response of a sporadic channel and to buffer an incoming message while waiting for the host to begin accepting data (latency problem).

3. Utilization of FIFO Buffer

While data and control tokens on the ring move in one direction only, information will go both ways through the adapter. After having decided that the adapter should incorporate a FIFO buffer, it was realized that it could beneficially be used handling data in both directions. To accomplish this a multiplexor was chosen and, by means of the microprogram, input to the FIFO Buffer is switched to the right paths. (For reference see Figure 3.) On the output end of the buffer no such switching was necessary, since either the PDA or the RI would be signaled for whom data is ready on the data out lines.

To enable the Adapter to have two sequential data bytes available, in parallel, to be released to the PDA as a 16-bit word, an eight-bit buffer is placed onto the out lines of the FIFO Buffer, into which one byte is locked (latched) while the other is made available in parallel.

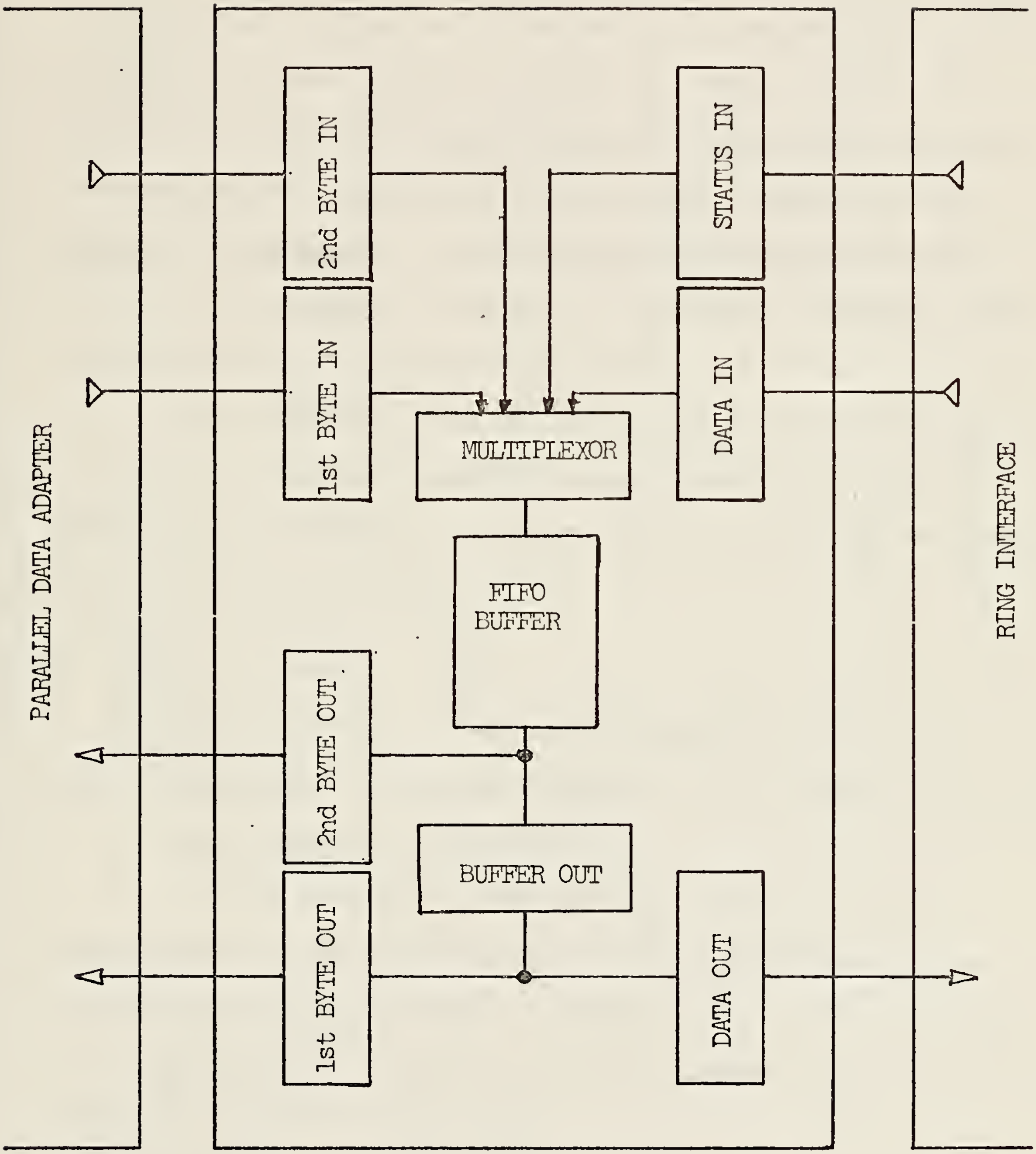


Fig. 3. Block Diagram of Ring Interface/Host Adapter (Microcontroller not shown)

4. <u>The RIHA</u>

Information about the actual design of the RIHA is contained in Figures 12 through 20. As mentioned above, 12 through 14 are taken from Ref. [1] which treats the basic version of the microcontroller while figure 15 shows the circuitry of the added Jump External (JEX) Feature. Figures 16 through 20 contain information about the RIHA. The circuitry shown in figures 18 and 19 is actually found on one board as seen from figure 16.

All external connections of the RIHA are indicated on figure 18 and all internal connections to the microcontroller on figure 19. Figure 20 shows the pin assignment for internal connection.

B. THE PDA INTERFACE LINES

The PDA Interface lines are discussed in detail in Refs. [8] and [9]. The main points will be brought out here.

1. <u>Data Lines</u> (see figure 4)

In its basic form (which will be used at this installation), the PDA supports 16 lines for output data and the same number of lines for input data. In each case a seventeenth line is provided for transfer of a parity bit but not utilized at this point.

2. <u>Control Lines</u> (see figure 5)

<u>Write Select</u> and <u>Read Select</u> are lines which are raised by the PDA if the RIHA has been selected for a write-type or read-type operation respectively. Either line will stay up until the operation is completed.

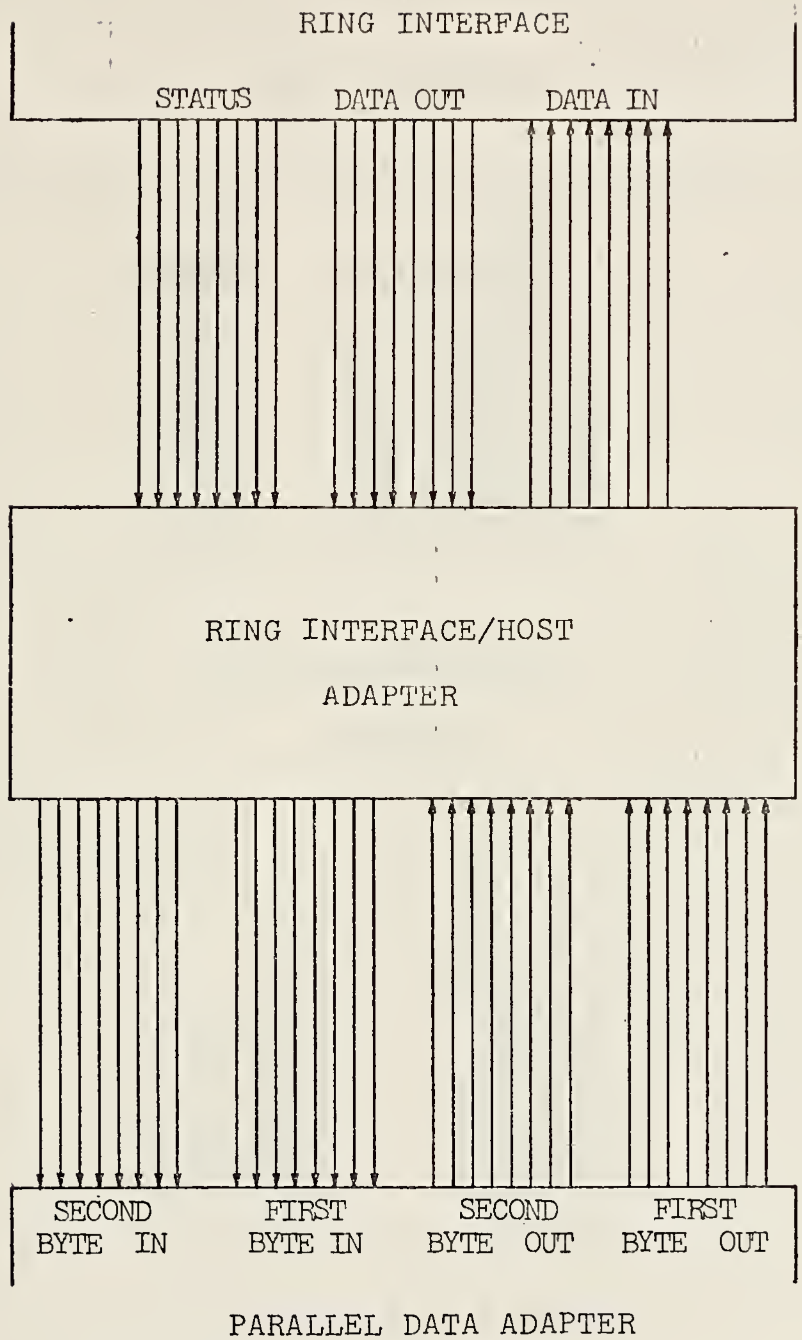


Fig. 4. Interface Data Lines

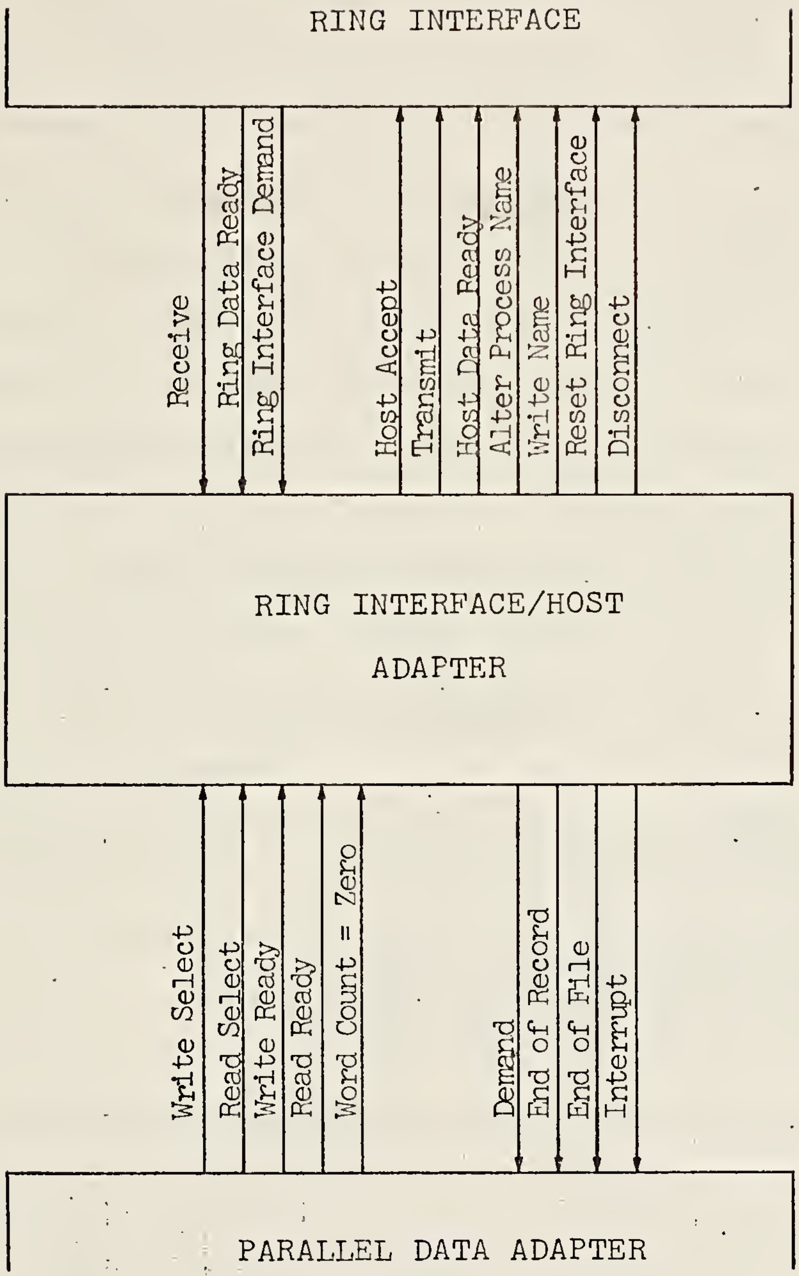


Fig. 5. Interface Control Lines

The Write Ready line is raised in a write operation and it notifies the RIHA that the next data word is ready on the Output Data Bus. The RIHA may react in the following ways: Raising the Demand line means the data word has been accepted. Raising End of Record (EOR), End of File (EOF), or Interrupt also resets the Write Ready line, their interpretation will be discussed later.

The Read Ready line is raised in a read operation and signifies that the PDA is ready to take the next word of data over the Input Data Bus. In this context raising the Demand line tells the PDA that the next data word is ready on the PDA Input Data Bus. Raising of EOR, EOF, or Interrupt again resets the Read Ready line, but their interpretation will be discussed later.

The line Word Count equals 0 (WC = 0) is raised by the PDA to indicate

in a write operation:	the channel has no additional data (normal ending of a write operation)
in a read operation:	the channel will not take any more data (if this happens during a Receive Sequence it indicates an error condition and is treated as such).

In both cases the PDA expects EOR, EOF, or Interrupt to be raised by the RIHA.

EOR and EOF both indicate that the RIHA has completed its operation and will not generate or accept any additional data. As a reaction to either, the PDA presents Channel End and Device End to the channel. With EOF, in addition

to the above, Unit Exception Status will be presented, which can be used as an indication to the host software of any error that may require a Status Request Message from the CPU for more detailed information.

The Interrupt line is raised by the RIHA to preempt a Transmit Sequence. When the host had raised WS and WR and the RI is waiting for the ring to become available for transmission, but a message for this host is detected on the ring, then Interrupt is raised to advise the host to drop its request for a Transmit Sequence for a moment and issue a Read Command to first handle the incoming message.

Two further control lines supported by the PDA, Redundancy Error and Suppress Parity Error, are not utilized by the RIHA at this time.

C. THE RING INTERFACE CONNECTIONS

1. Data Lines (see figure 4)

For data transfer between the RIHA and the RI an 8-bit data bus is provided in each direction. During a Receive Sequence data is made available by the RI on one bus and then the RIHA is informed that it may receive it. When the RI signals during a Transmit Sequence that it is ready for the next byte, data is put by the RIHA onto the other bus and the RI is notified that host data is ready for delivery.

2. Control Lines (see figure 5)

In figure 5 the control lines are graphically grouped according to the direction in which information is conveyed. Another way to group them on a functional basis is the following:

Receive Group (lines used during a Receive Sequence)

RECEIVE	(RCV)
RING DATA READY	(RDR)
HOST ACCEPT	(HA)

Transmit Group (lines used during a Transmit Sequence)

TRANSMIT	(XMIT)
RING INTERFACE DEMAND	(RID)
HOST DATA READY	(HDR)

Local Command Group (lines used in reaction to a Local Command Message)

ALTER PROCESS NAME	(APN)
WRITE NAME	(WRTN)
RESET RING INTERFACE	(RESET)
DISCONNECT	(DISC)

a. The Receive Group

RCV (from RI to RIHA)

Raising this line notifies the RIHA that a message for a process residing on this host is coming in from the ring. This logically puts the RIHA into the Receive Sequence. If RCV is raised while the RIHA is in a Transmit Sequence (waiting for the ring to become available for transmission) it immediately notifies the host that it is going to abort that sequence and will switch to the Receive Sequence. The RCV line is only lowered after the last byte has been transferred to the RIHA.

RDR (from RI to RIHA)

Raising this line indicates that the next (or the first) data byte is ready on the data bus to be received by the RIHA. After the last data byte has been transferred to the Adapter and RCV is lowered, the significance of RDR is redefined as: Status Byte valid. RDR is never lowered until HA is raised to preserve an interlocked "handshaking" mode of operation.

HA (from RIHA to RI)

Raising this line implies that data from the bus has been received. This causes RDR to fall. After transfer of the last byte of data and lowering of RCV, HA is redefined as: Status Byte has been received. It is raised after RDR shows: Status Byte valid. This causes RDR to fall again allowing HA to fall.

b. The Transmit Group

XMIT (from RIHA to RI)

Raising this line indicates that the host wants to transmit a message onto the ring. It stays up until the last data byte has been delivered to the RI or until a raised RCV indicates preemption of the Transmit Sequence by an incoming message. Preemption will only occur before the first byte has been requested by the RI.

RID (from RI to RIHA)

The first time this line goes up after XMIT has been raised it implies that the ring became available for transmission.

It also notifies the Adapter that the RI is asking for the delivery of a data byte. After the last data byte was delivered and XMIT has been lowered RID is redefined to: Status Byte valid. RID is lowered after HDR was raised and the data byte was taken off the bus.

HDR (from RIHA to RI)

This line is raised when the RI had asked for the next data byte and this byte is ready for delivery on the data bus. It allows RID to fall. After the last data byte was delivered and XMIT has been lowered, HDR is redefined to: Validity of Status Byte has been recognized. This allows RID to fall. HDR is always lowered in reaction to the drop of RID.

c. Local Command Group

APN (from RIHA to RI)

Raising this line indicates that a Local Command Message has been received from the host which either instructs the RI to delete a name from its list of process names or to insert a new name, depending on the state of the WRTN line. After APN has risen no change in WRTN is allowed.

WRTN (from RIHA to RI)

If this line is down, then the meaning of APN is: delete the process whose name is on the data bus. If this line is raised then the meaning of APN is: insert the process whose name is on the data bus into the list of valid process names. Raising RID allows first WRTN and then APN to drop, which in turn causes RID to go down.

DISC (from RIHA to RI)

Raising of this line indicates that a Local Command Message has been received from the host which instructs the RI to disconnect from the ring. The RI may wait for an appropriate moment to disconnect; whether it is connected or disconnected is indicated at all times by the respective bit in the Status Byte which can be asked for by the host issuing a Status Request Message. Lowering of DISC lets the RI switch back into the ring.

RESET (from RIHA to RI)

This line is used for two purposes:

1. During the power-on procedure of the RI its micro-controller is put to the start of its program by raising this line.

2. During a Transmit Sequence; when the host ties up the ring for too long a time the RI will automatically disconnect from the ring and free it for other messages. The only way to switch the RI back into the ring is by raising this line first and then sending a Local Command Message to get the RI connected again.

d. The Status Byte (8 lines from RI to RIHA)

The Status Byte consists of 8 bits which represent information about the state of the RI. Their significance is:

Receive Group

S_0 - CRC Error

S_1 - Overrun

Transmit Group

S_2 - CRC Error
S_3 - Overrun
S_4 - MSB1
S_5 - MSB2

Miscellaneous

S_6 - Ring Error
S_7 - Disconnected

For more details on these see Ref. [6].

The Status Byte is used in different ways according to the sequence that the RIHA is executing:

Receive Sequence

After a message from the ring has been received and transferred to the host, the RIHA waits until the RI declares the Status Byte to be valid and then appends one more 2-byte word consisting of the Status Byte and a byte of zeros. The same is done if the ring were that much faster than the PDA to cause a Receive Overrun Error. In this manner the receiving program inside the host gets all the RI status information concerning that message.

Transmit Sequence

After a message has been transmitted onto the ring the RIHA waits for the RI to declare the Status Byte to be valid (after the message has circulated around the ring), and then tests the Transmit Group to decide whether the message went around the ring without errors. If this is the case, it

raises EOR, otherwise it raises EOF, which in addition to Channel End and Device End lets the PDA present Unit Exception to the channel. In this manner the transmitting program is informed whether the message correctly reached its destination or has to be retransmitted. This information about what went wrong is acquired by sending a Request Status Message from host to RIHA.

D. THE FIFO BUFFER

The size of the FIFO buffer's memory was chosen to be 1024 x 8 bits. It was designed to act as a "Fall-Through Buffer." This means the first data that enters the buffer seems to fall through and is immediately available on the output side. This was accomplished in the following way (for reference see Figure 6):

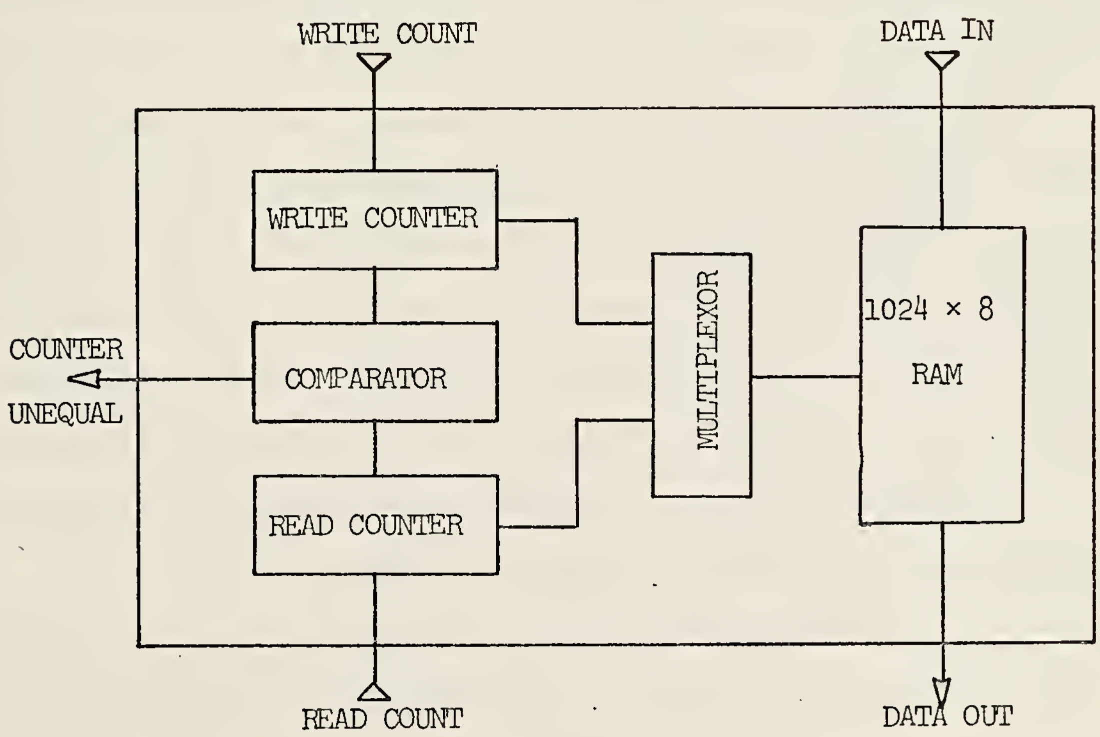


Fig. 6. FIFO BUFFER ARCHITECTURE

One 10-bit counter is used as a pointer to the memory location next to be written into and another 10-bit counter serves as pointer to the location from which to read the next data byte. At the start both show zero, i.e., they point to the same storage location. Therefore equality of pointers implies an empty memory as long as nothing is read from or written into memory. After each Read/Write operation the respective counter is incremented and hence points to the next cell to be read from/written into. Should the "Writes" come faster than the "Reads," at some time (possibly after several wrap-arounds) the Write Counter (WCNT) will point to the cell which is also the next to be read from. Therefore equality of counters after a Write operation indicates an Overrun. On the other hand, if one or more "Writes" had been previously executed, (i.e. the FIFO was partly filled) equality of Read Counter (RCNT) and Write Counter (WCNT) would imply: the "Reads" caught up with the "Writes." The FIFO would be empty and the next operation has to be a Write. To detect these various conditions a 10-bit comparator was built, the result of which is true as long as the counters point to different locations. It is false after resetting both counters to zero at the start of any message sequence as a measure to "erase" any buffer content.

If after each Write or Read operation in the RIHA program, the related counter is incremented (which forces the Counter-Not-Equal (CNTUNEQAL) line up) and care is taken

that at each start of a new sequence a "Write" is executed first, then the following must be true:

A drop of CNTUNEQAL indicates after a

WRITE: WCNT has wrapped around and caught up with RCNT: Overflow of FIFO Buffer

READ: RCNT caught up with WCNT: FIFO Buffer is empty.

E. THE MESSAGE FORMAT

Messages received by the RI from the ring for its host are of no further concern to the RIHA. They are transferred to the RIHA as 8-bit bytes one at a time, written into the FIFO Buffer and later read from there to be prepared for release to the host two bytes at a time as 16-bit words. For more detail about ring message formats and protocols see Ref. [6].

In the other direction, two types of messages have to be discernible. A Local Command Message (LCM), which is an instruction or request from the host to the RI and has to be interpreted by the RIHA, and the regular Transmit Message (TM) to be sent over the ring. The LCM is required to consist of two bytes where the first byte indicates the type of LCM while the second may be used to supply additional data. On the other hand, each TM sent by any host onto the ring carries as its first two bytes the destination process name and the source process name. Therefore even the shortest possible message of this type consists of more than two bytes. This fact is taken advantage of to distinguish between LCM and TM as described below.

The PDA raises WS to indicate that it wants to send a message. Then WR is raised to signal the RIHA that the first 16-bit word is ready on the data bus to be accepted by the RI. After writing these first two bytes into its FIFO Buffer the RIHA acknowledges acceptance by raising Demand. This allows WR to drop. After transfer of the last two bytes WC = 0 is raised together with WR to inform the RIHA that the CPU has no more data to transfer. Consequently WC = 0 will not be raised after the first two bytes of a TM, or expressed the other way: if WC = 0 goes up after the first two bytes being transferred, then the message is an LCM.

Figure 7 shows which types of messages are at this time identifiable by the RIHA's program.

Insert Process Name	WRITE	NAME	
Delete Process Name	CLEAR	NAME	
Disconnect from Ring	DISCONNECT		
Connect to the Ring	CONNECT		
Reset RI Microcontroller	RESETRI		
Status Request Message	STATREQU		
Transmit Message	DESTINATION	SOURCE	TEXT BYTE 1

Fig. 7. MESSAGE FORMATS

F. THE MICROCONTROLLER

1. General Description

The microcontroller, which represents the heart of the Ring Interface/Host Adapter (RIHA), was designed at this school for various similar applications by Assistant Professor Raymond H. Brubaker, Jr., with Mike Harris, whose thesis topic was the development of the Ring Interface. A detailed description of the microcontroller will be found in Ref. [1], but the main features are reviewed here. Taken from that reference and included in this text as Appendix A is a block diagram of the microcontroller's architecture (Fig. 12), its instruction format (Fig. 13), the microcontroller's circuitry (Fig. 14), and the added JEX feature circuitry (Fig. 15).

The microcontroller's instruction set consists of five instructions:

Output	(OUT)
Jump Unconditional	(JU)
Jump on True Input	(JT)
Jump on False Input	(JF)
Jump on External Input	(JEX)

An OUT instruction displays data supplied by its lower 8 bits on an 8-bit data out bus and then selects one out of up to 32 output lines and concurrently strobes it for a 100 nanosecond time interval.

On a JU instruction the program branches to any location of its available memory that is specified in the lower 13 bits of the instruction.

A JT or a JF instruction selects one out of up to 32 input lines for a test. If the line is up with a JT or down with a JF instruction, then the program branches to the location on the same page that is specified in the 8 lower bits of the instruction. Otherwise the next sequential instruction is executed (with fall-through to the next page possible).

The JEX instruction was added to the basic microcontroller for its application in the RIHA. A drawing of the circuitry is included as figure 15. On a JEX command an unconditional jump occurs to an address specified in the instruction with the four low order bits modified by an outside source. In this application bits 4 through 7 are extracted from the first byte of an incoming LCM and used to differentiate between the possible message types.

Using these five instructions a program may be written whose flow can be varied according to up to 32 input variables and which generates a sequence of output signals that select one of up to 32 "devices" with data displayed on the out bus to further control these devices.

2. The RIHA Microcontroller Program

a. The Language

To ease programming and debugging of the Microcontroller an assembly language called SMAL was created and an assembler was written in PLM [4] by Assistant Professor Gary A. Kildall [2]. The assembler runs on the Intellec 8 or Intellec 80 developmental system [5].

As an aid to reading a program written in SMAL, the operators used in the language are reviewed here. For more detail see Ref. [2].

Value Definition

Operator: =

Example: UP = 1

Assigns a value to an identifier.

Unconditional Jump

Operator: =:

Example: =: RECEIVE

The identifier to the right of the operator represents an address for an unconditional jump to anywhere in the available memory.

Jump External

Operator: =::

Example: 0 =:: JEXTABLE

The zero is just a placeholder. JEXTABLE is an address in memory whose last four bits are zero. Since these four low order bits are replaced when the instruction is executed, an unconditional jump to one out of 16 sequential locations in memory occurs. If a sequence of JU commands is found in these locations the effect is that of a "case statement."

Conditional Jump

Operator: =:

Example: RDR =: RECEIVE

The identifier to the left of the operator represents one of 32 possible input lines, which is tested and if the test returns true, a jump to the address indicated by the identifier to the right of the operator is executed. This address has to be on the same page in memory as the conditional jump instruction. The above mentioned test returns "True" if either the line tested is up or, with a minus sign in front of the line name (-RDR =: RECEIVE), when it is down, otherwise the test returns "False".

Note: * to the right of a jump operator (=:) indicates looping on that instruction.

Example: RDR =: *

The loop is exited when RDR goes down.

Output Statement

Operator: :=

Example: SEL41 := RIDATA

The identifier to the left of the operator specifies one out of 32 possible "devices". The identifier to the right represents data which is displayed on the out bus, while the indicated device is strobed for a 100 nanosecond time interval.

Any line starting with a "/" is considered to be a comment line and disregarded by the assembler.

b. The Program Logic

Both the RIHA program and a set of flowcharts are included at the end of this thesis. The program with its flowchart is structured according to its functions with each function assigned a number shown at the entry points of the flowchart pages and as a comment line in the program.

Figures 8 through 11 show graphs in which the vertices contain the function number and represent the functions and the directed edges (arrows) denote possible transfer paths from a function to another according to specific decisions made at the function.

In the following paragraphs these functions will be interpreted. The flowchart page with the respective function number at its entry point should be used as reference.

0 START — The program idles in this part. Its attention may be called by either the PDA (transfer to 2) or by the RI (transfer to 10). Before starting a Receive Sequence, the issue of a Read Command by the channel may be requested by raising the Interrupt line.

2 INTERPRET — This "function" determines, whether the host wants to send a Local Command Message (transfer to 30) or a Transmit Message (transfer to 20).

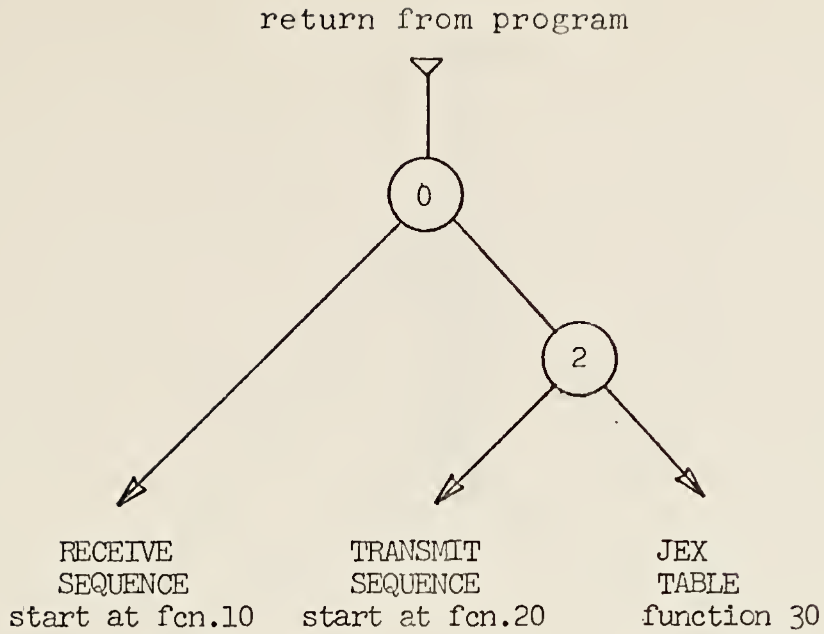
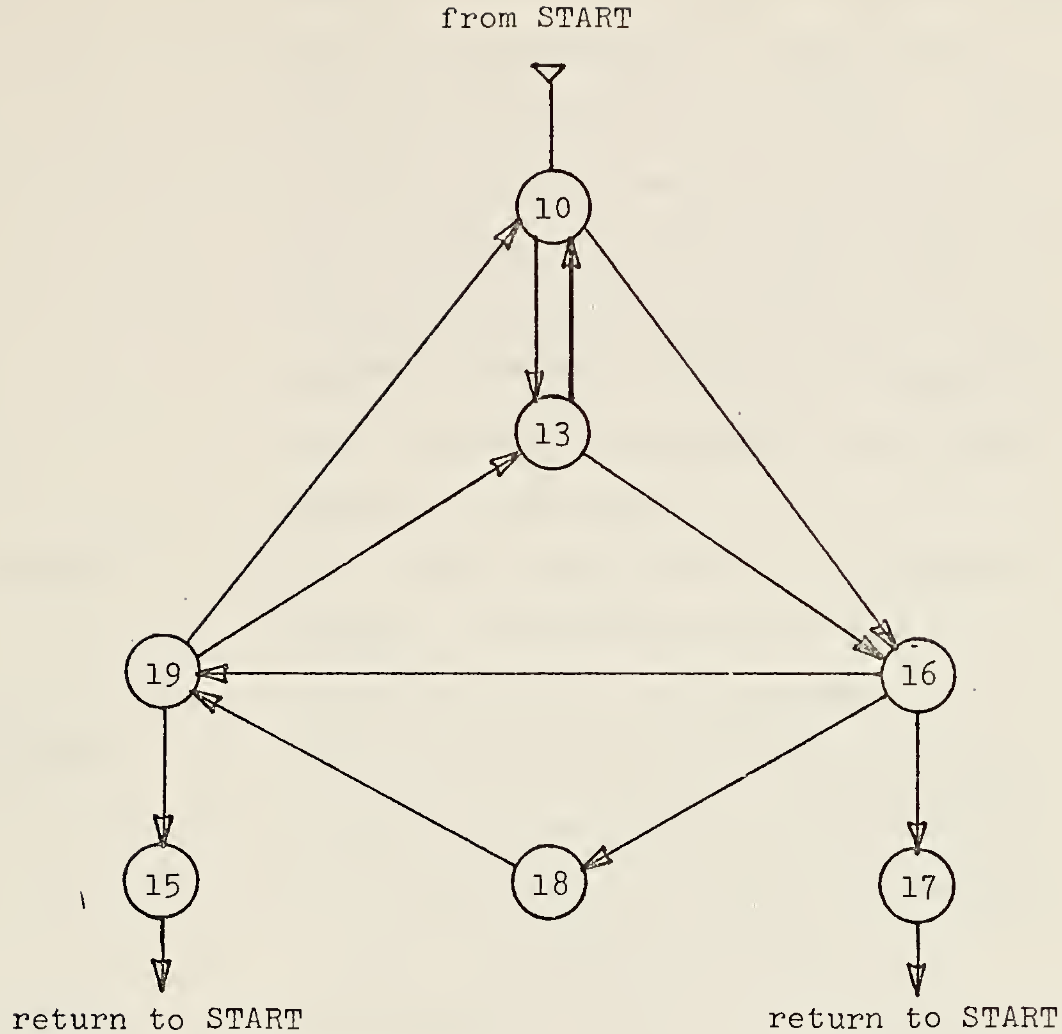


Fig. 8 . Directed Graph of Sequence Initiation



10 RECEIVE
13 SELECT
15 ADDSTATUS
16 RLSONE
17 RECOVER
18 RCVSECOND
19 RLSTWO

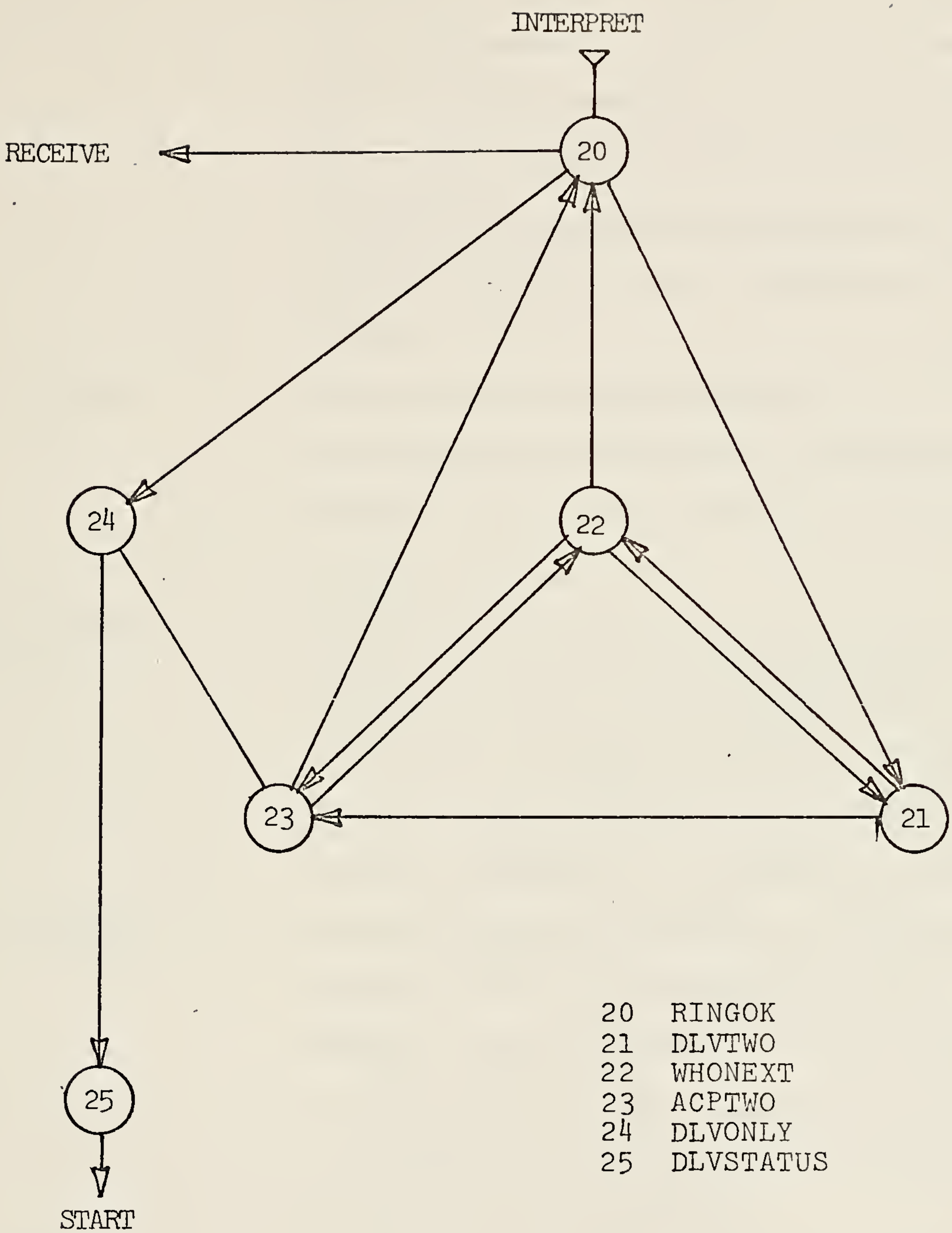
Fig. 9. Directed Graph of Receive Sequence

10	RECEIVE	This part is entered after the RI has indicated that it has ready the next data byte on the bus (RDR↑). This byte is received and the FIFO is checked. If it is full, then the next operation has to be a release of a 16-bit word to the PDA, which is forced by a transfer to (16). Should an overflow at the RI result from this, it will be recorded in the Status Byte by raising ROVR.
13	SELECT	This is the central loop of the Receive Sequence; either the RI (10) and the PDA (16) may call for service.
16	RLSONE	One byte is locked into the Out Buffer. If that empties the FIFO buffer, receipt of a second byte is forced by a transfer to (18). Otherwise go to (19).
19	RLSTWO	Two bytes are ready for the PDA and are released. If more data in FIFO, transfer to (13). Otherwise check whether message ended, then transfer to (15) or force a Receive by a transfer to (10). Note: The rise of RDR after RCV dropped is redefined to: Status Byte is valid.
18	RCVSECOND	Only entered from (16) if a second byte is needed to form a 16-bit word for the PDA. If the end of the message was reached

(RCV↓) a zero byte is written into the FIFO, otherwise one byte is received from the RI. No FIFO check is necessary since it had to be empty to get here.

15 ADDSTATUS — Entered from (19) after message end. FIFO is empty, Status Byte is valid and loaded into FIFO followed by a zero byte. Both are made available to the PDA as the last 16-bit word, then EOR is raised, which causes the PDA to signal channel end and Device End Status to the I/O channel.

17 RECOVER — Entered only if a Receive Sequence is interrupted by the host by raising WC=0 before the whole message was through. The RIHA causes a Receive Overrun (ROVR) in the RI by waiting on RCV to fall.



20 RINGOK
21 DLVTWO
22 WHONEXT
23 ACPTWO
24 DLVONLY
25 DLVSTATUS

Fig. 10 . Directed Graph of Transmit Sequence

20	RINGOK	After a Transmit Sequence is requested by the PDA and the RI informed of it (XMIT↑) the program waits in this loop for the ring to become available (first rise of RID). This Transmit Sequence may be preempted by an incoming message destined for the host (RCV↑) and a transfer to (10) occurs.
21	DLVTWO	Two bytes are delivered to the RI. If this empties the FIFO the acceptance of a 16-bit word from the PDA is forced by transfer to (23).
22	WHONEXT	Central loop of the Transmit Sequence; either the RI(21) or the PDA(23) may call for service. A test is made for Transmit Overrun at the RI which cancels this Transmit Sequence by a transfer to (20).
23	ACPTWO	Entered after PDA raised WR; two bytes are accepted. WC = 0 up indicates end of message, transfer to (24). If the FIFO is full a delivery of two bytes is forced by a transfer to (20) and (21) to make room for the next PDA word.
24	DLVONLY	If entered from (20): Transmit Overrun has occurred, XMIT is taken down to redefine RID to: Status Byte is valid. If entered from (23): The rest of the message is delivered to the RI; then transfer to (25).

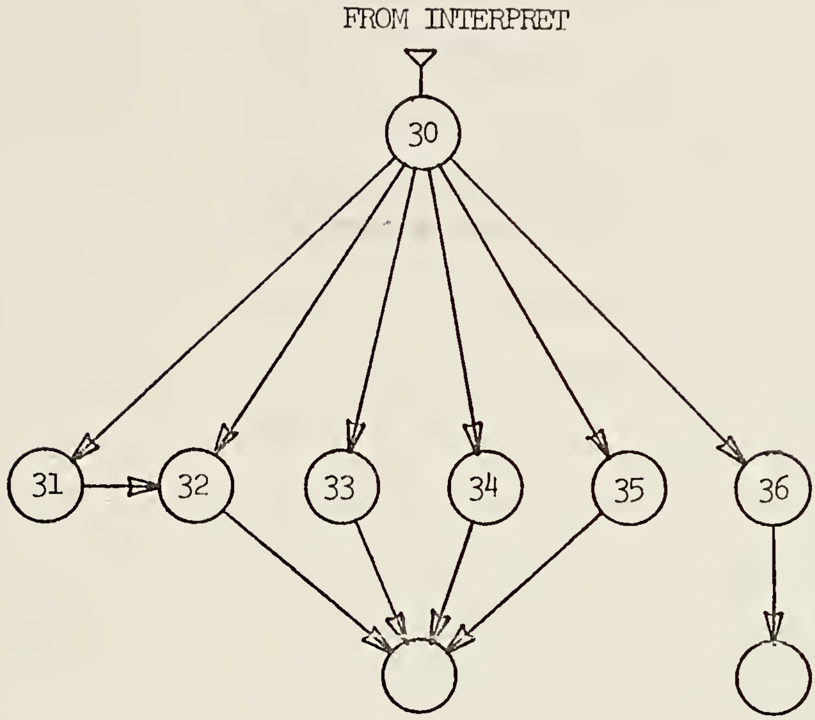
25 DLVSTATUS

Program is looping on redefined RID, waiting for the Status Byte to become valid; then the Status Byte is examined by the RIHA for correctness and according to the result either EOR or EOF is raised.

EOR indicates to the host:

Message transmitted and correctly received at destination.

EOF indicates: Something went wrong, issue a Status Request Message to get further details.



30	JEXTABLE	34	CONNECT
31	WRITENAME	35	RESETRI
32	CLEARNAME	36	STATREQ
33	DISCONNECT		

Fig. 11. Jump External

30	JEXTABLE	Here the JEX instruction is used to direct the program to the right program part according to the Local Command Message sent by the host.
31	WRITENAME	(See CLEARNAME)
32	CLEARNAME	The second byte of the Local Command Message containing the name of a process is handed to the RI. According to the state of the WRTN line, the RI deletes that name from (WRTN↓) or inserts it into (WRTN↑) its list of valid processes.
33	DISCONNECT	Raises the DISC line and waits on the Status Bit RDIS for reaction of the RI.
34	CONNECT	Lowers the DISC line and waits on the Status Bit RDIS for reaction of the RI.
35	RESETRI	Raises RST for a minimum of 1.1 micro-seconds.
36	STATREQ	Resets both FIFO counters. Causes a Read Command if not yet outstanding and transfers to (15) where the Status Byte is made available to the PDA.

V. RECOMMENDATIONS AND CONCLUSIONS

A. RECOMMENDATIONS

The next steps to be taken after testing RI and RIHA singly at low speeds, would be to combine them and program available MCS-8 microcomputers [5] to simulate the IBM Parallel Data Adapter on one side and the Ring on the other.

Internal improvements to the RIHA as seen by the author would include:

1. A "Time-up" Circuitry, adjustable to various time spans, that could be reset by the microcontroller with every strobe of one of its out lines. In case its preset time should elapse, a recovery procedure could be started and/or an indication to the outside could signal that the program got caught in an endless loop.
2. To enable evaluation of the device's performance, a number of counters could be strobed by the microcontroller, according to instructions to that effect placed at strategic points in its SMAL program.

B. CONCLUSIONS

The chosen approach, to modularize hardware and software, has proven to be of great advantage. Two devices, the Ring Interface [6] and the Ring Interface/Host Adapter (theses w· tten at the same time), were implemented using the same Microcontroller [1] and its language SMAL [2]. This provided

for better communication between all parties concerned and increased greatly the flexibility with respect to necessary changes.

The preliminary testing of the RIHA was done by manually setting the control lines to test the various sequences. According to its program and with an instruction cycle time of 1.1 microseconds the RIHA should be able to handle data in burst mode up to the following speeds:

From PDA towards RI	-Accept from PDA	106 kilobytes/second
	-Deliver to RI	129 kilobytes/second
From RI towards PDA	-Receive from RI	82 kilobytes/second
	-Release to PDA	113 kilobytes/second

It thus seems reasonable to assume that the RIHA could sustain a data rate of 50 kilobytes/second in both directions.

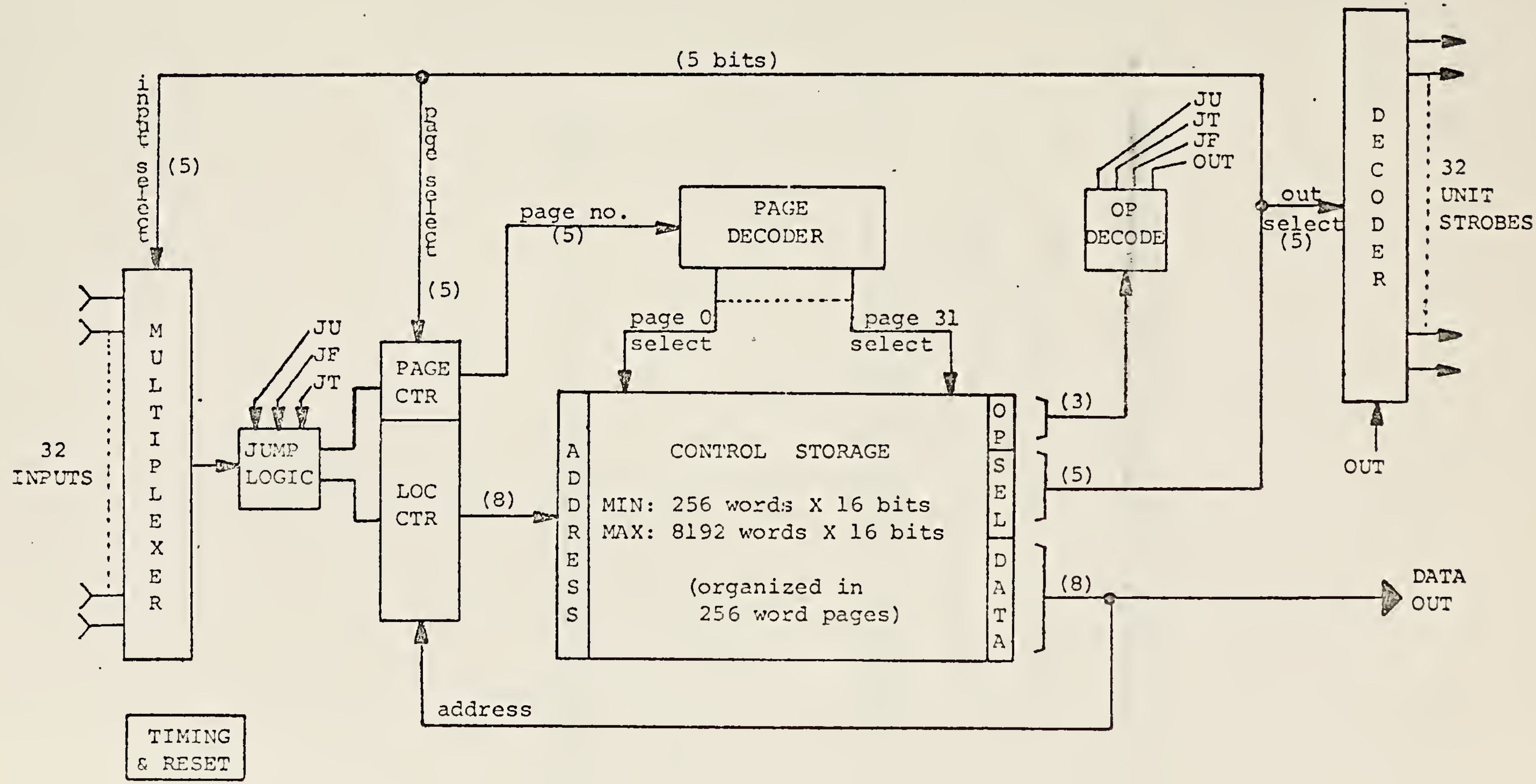


Fig. 12. Block Diagram of the Microcontroller Architecture

	15 — 13	12 — 8	7 — 0
OUT:	0 0 0	unit select	data out
JU:	1 0 0	page number	location
JT:	1 0 1	input select	location
JF:	1 1 0	input select	location

Fig. 13. The Microcontroller Instruction Format

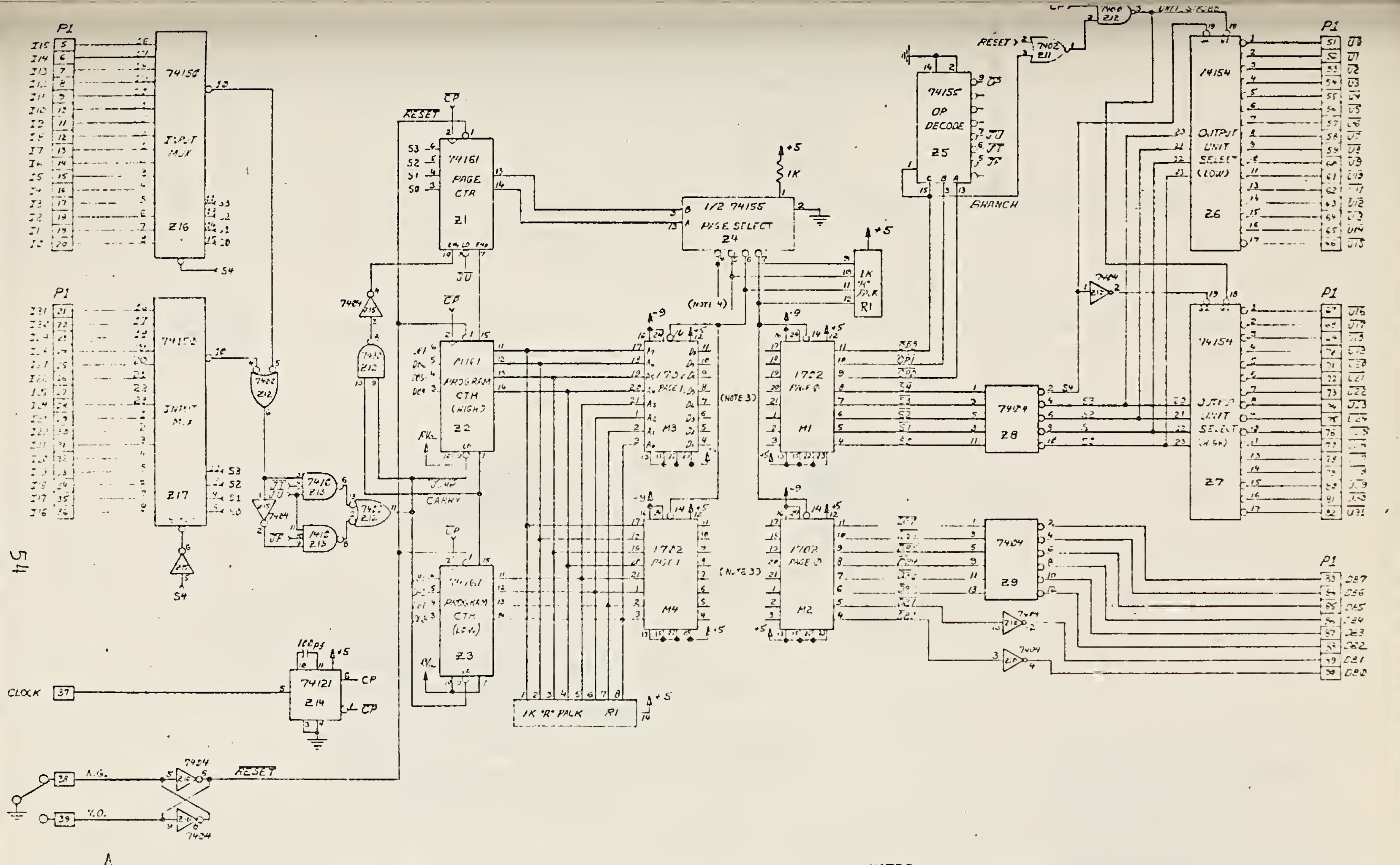


Fig. 14. The Microcontroller Circuitry

NOTES:

1. P1 IS 100 PIN EDGE CONNECTOR
2. CP IS 100 NS. (NOMINAL)
3. ALL ROM'S CONNECT TO COMMON ADDR. LINES. ALL PAGE 1 DATA LINES (AM4-11) WIRED TO CORRESPONDING PAGE 0 LINES.
4. 24 PINS 4 & 5 FOR EXPANSION TO 4-PAGE ROM.

NAVAL POSTGRADUATE SCHOOL

ROM-BASED MICROCONTROLLER

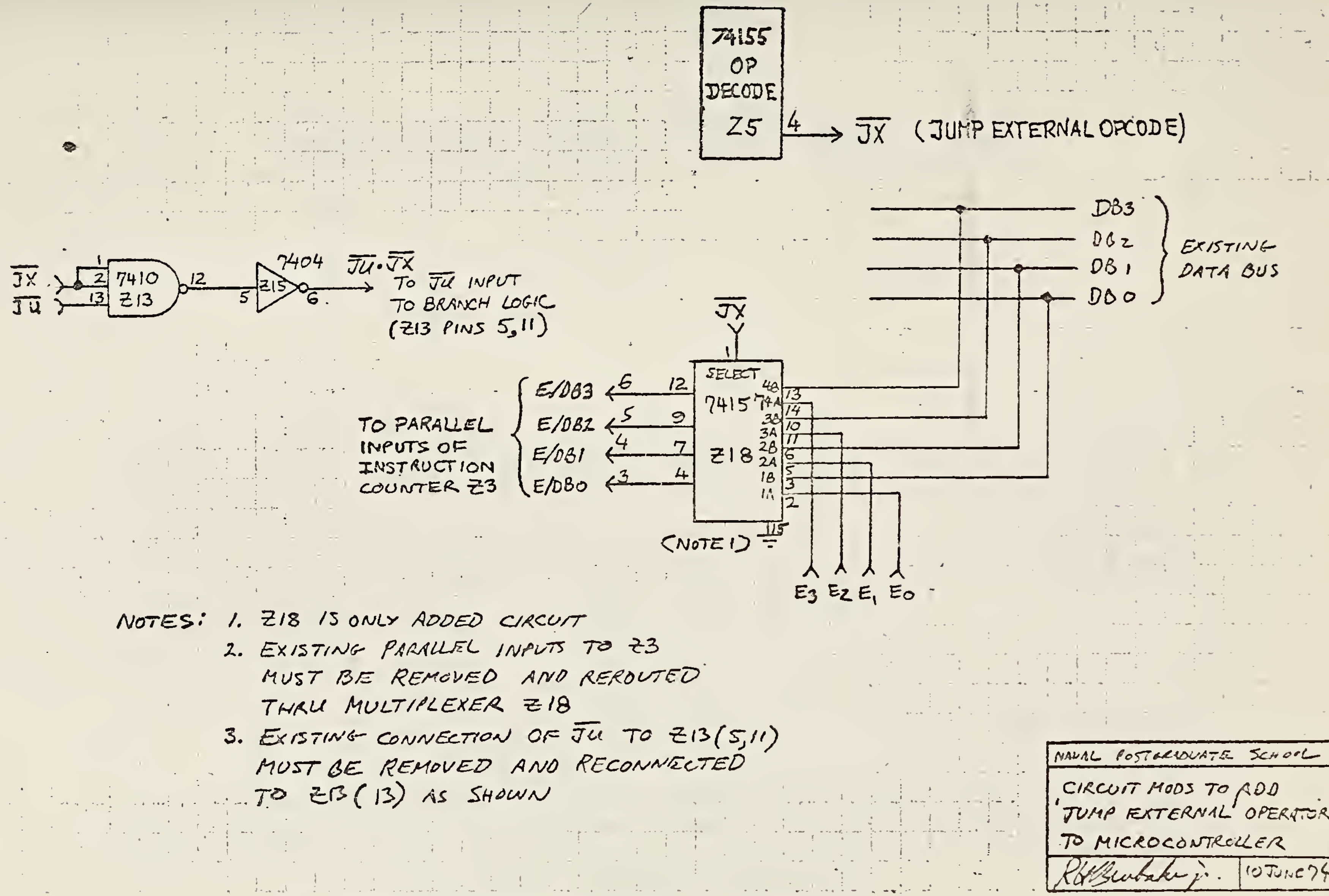


Fig. 15. Jump External Feature

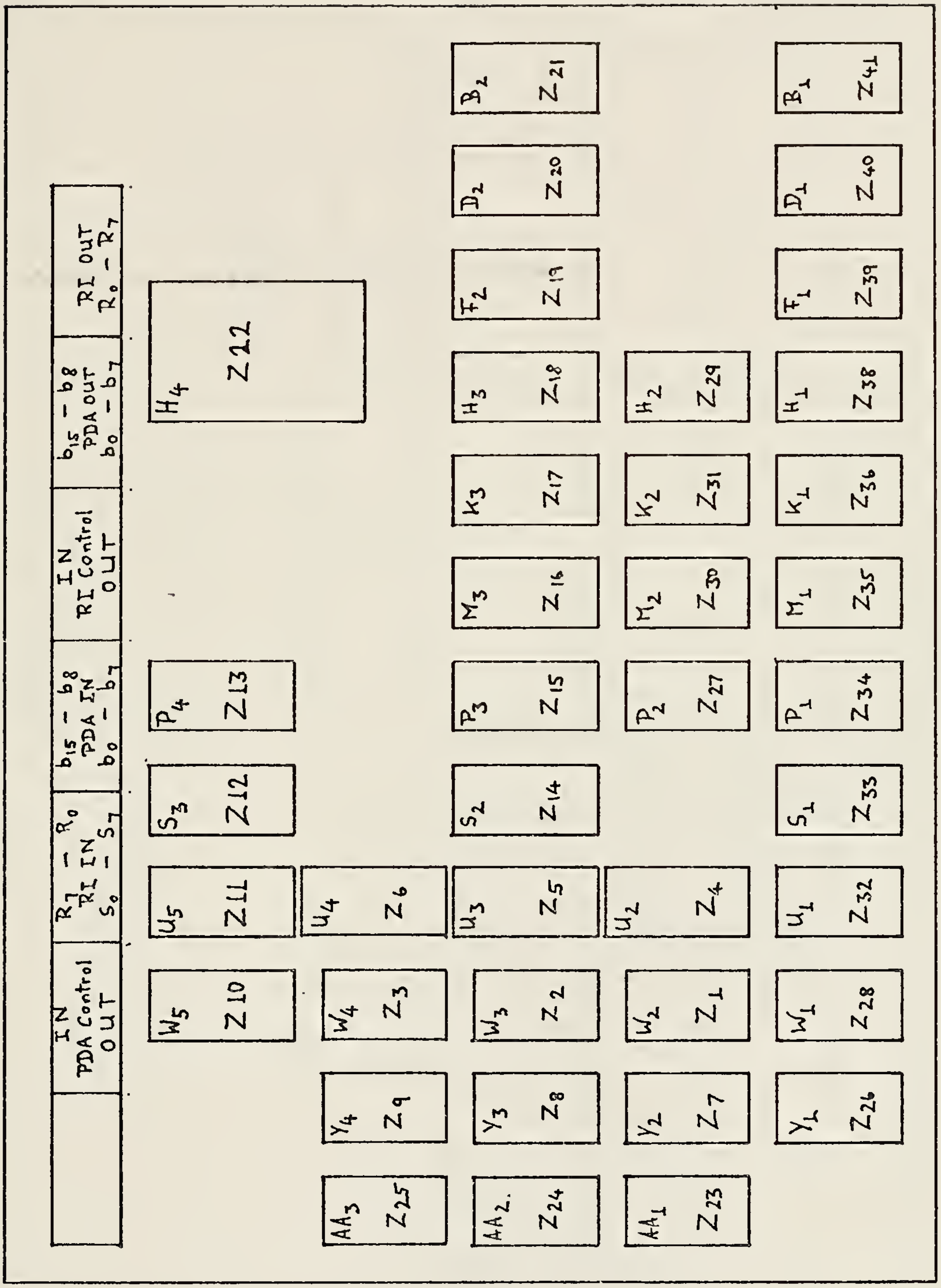


Fig. 16. Layout of RIHA Board

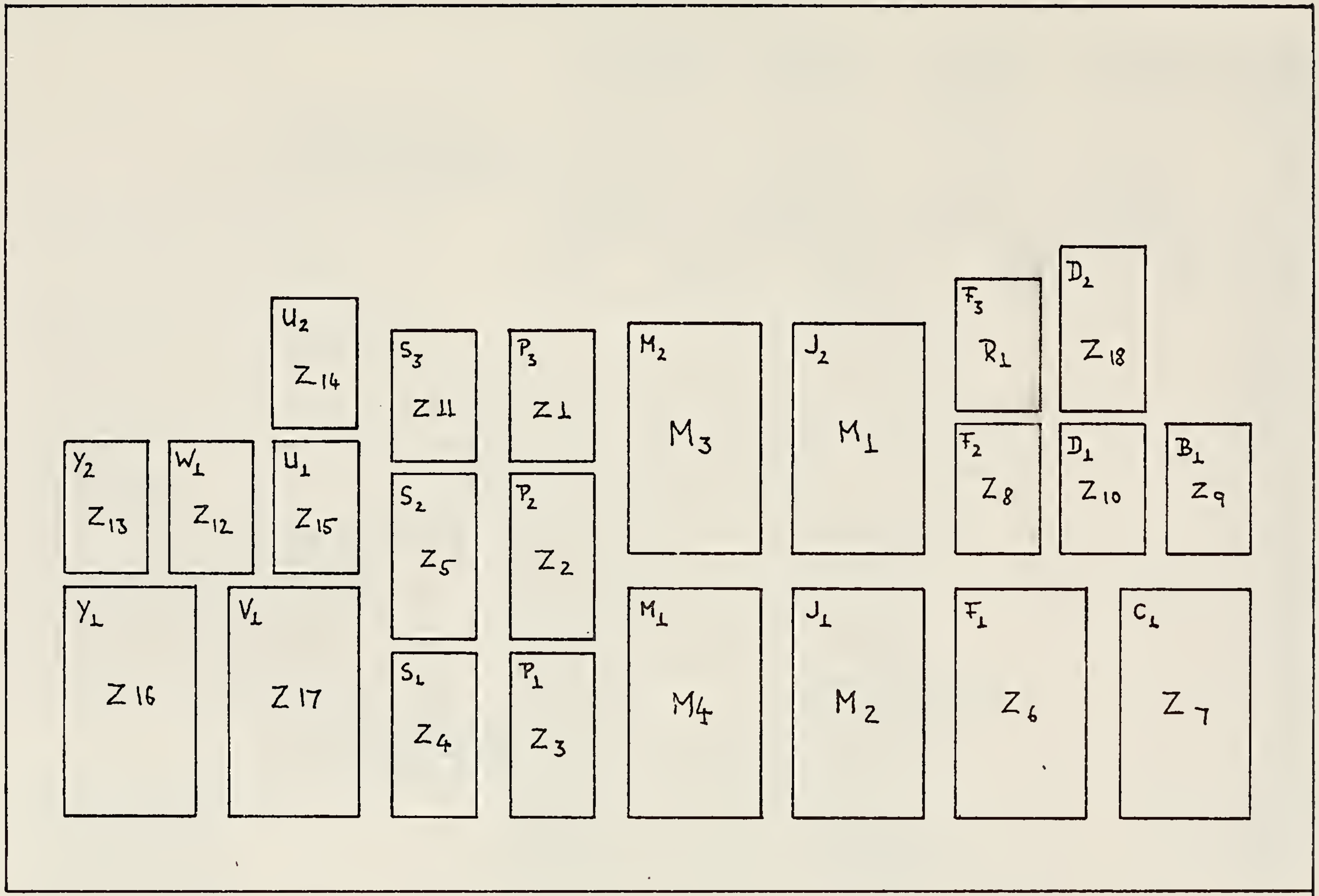


Fig. 17. Layout of RIHA Microcontroller

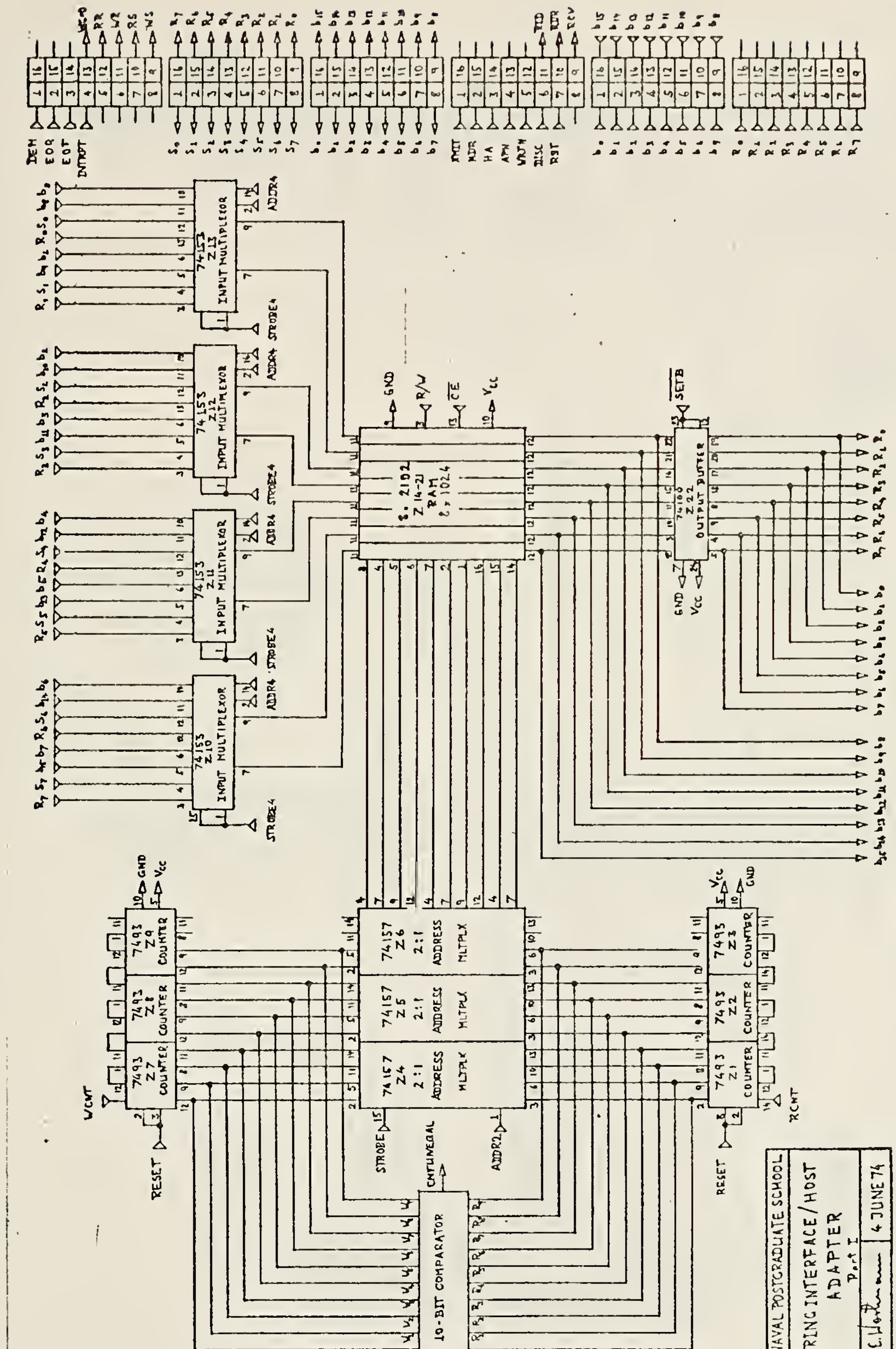


Fig. 18. RIHA Circuitry I

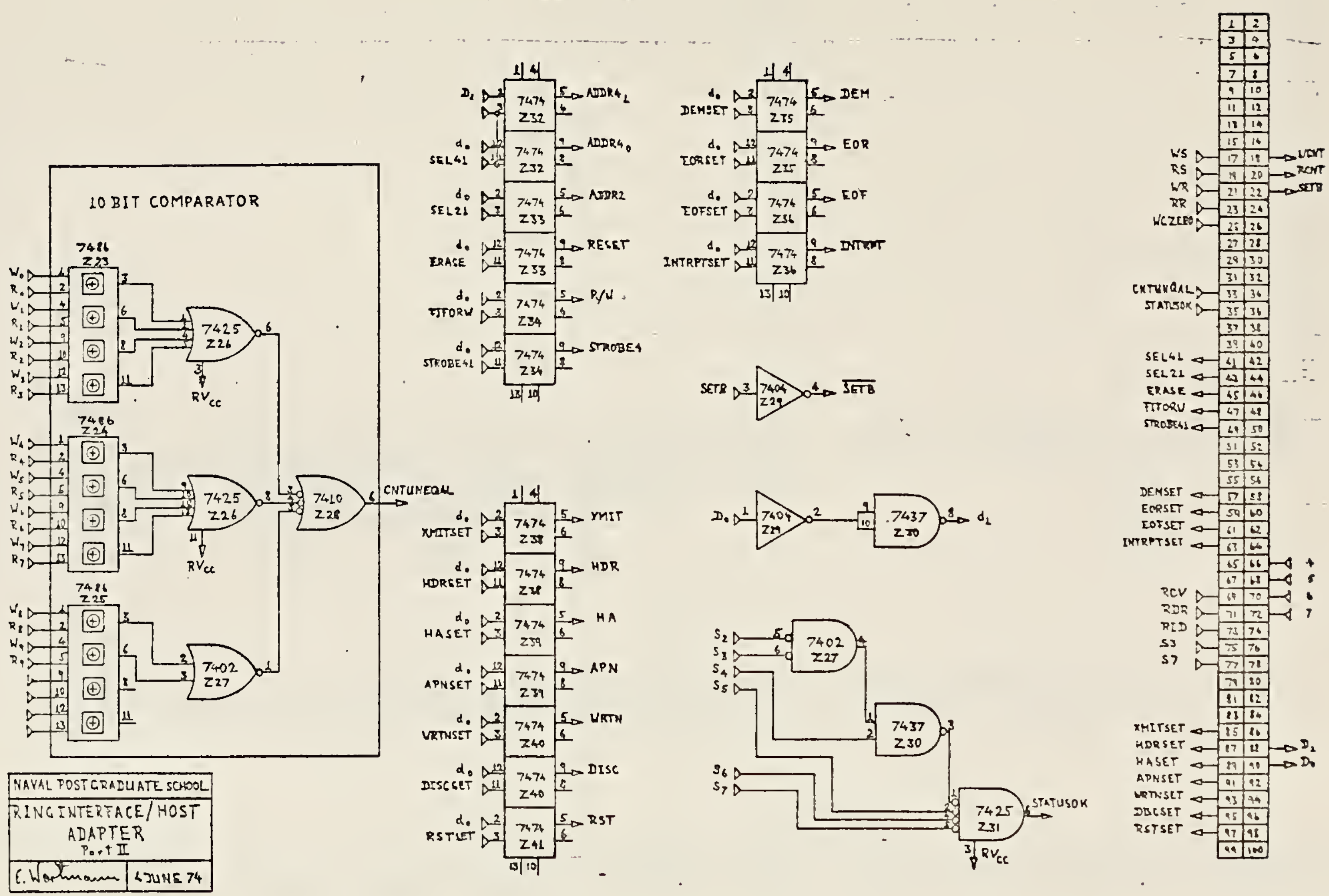
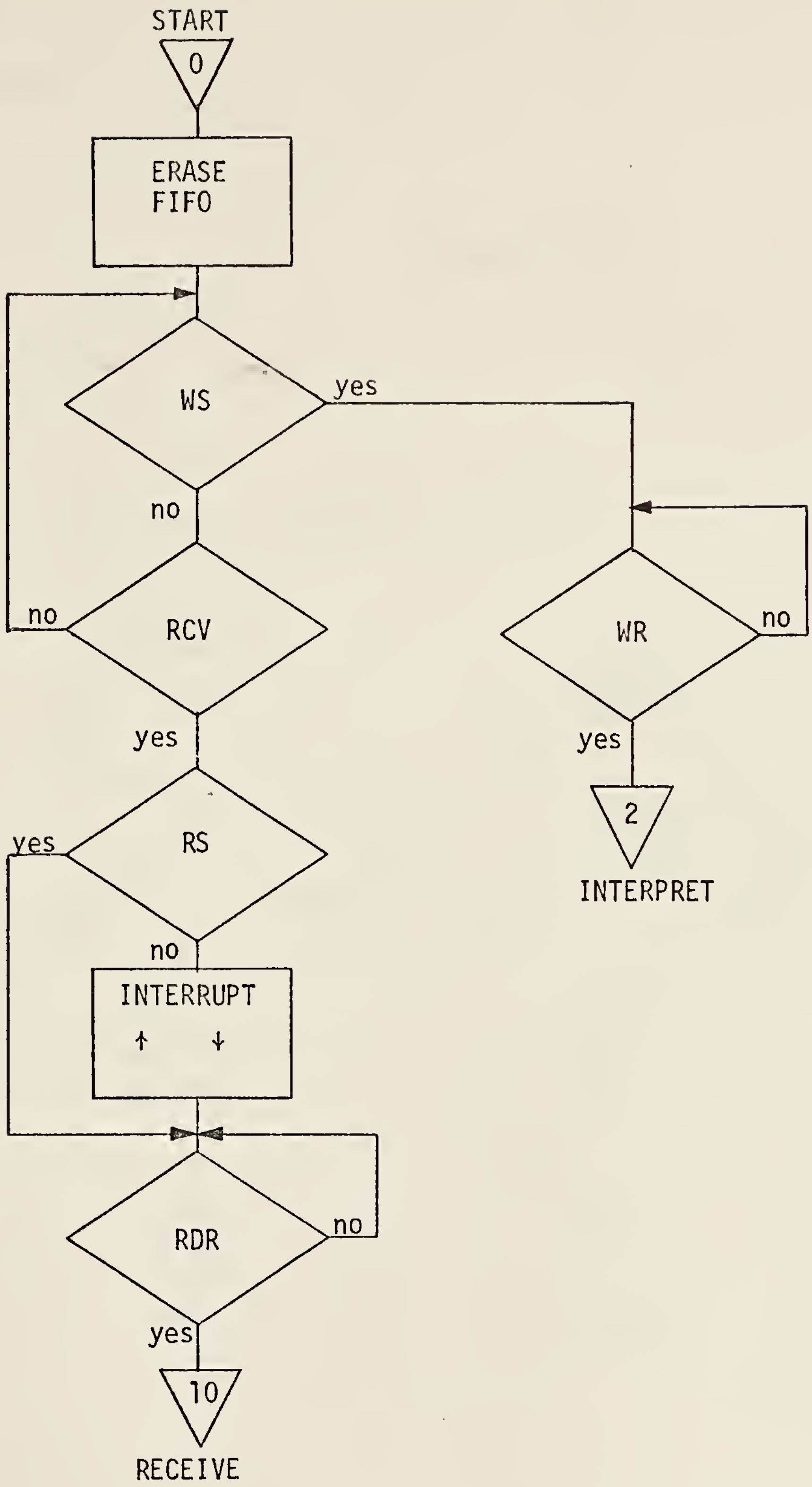
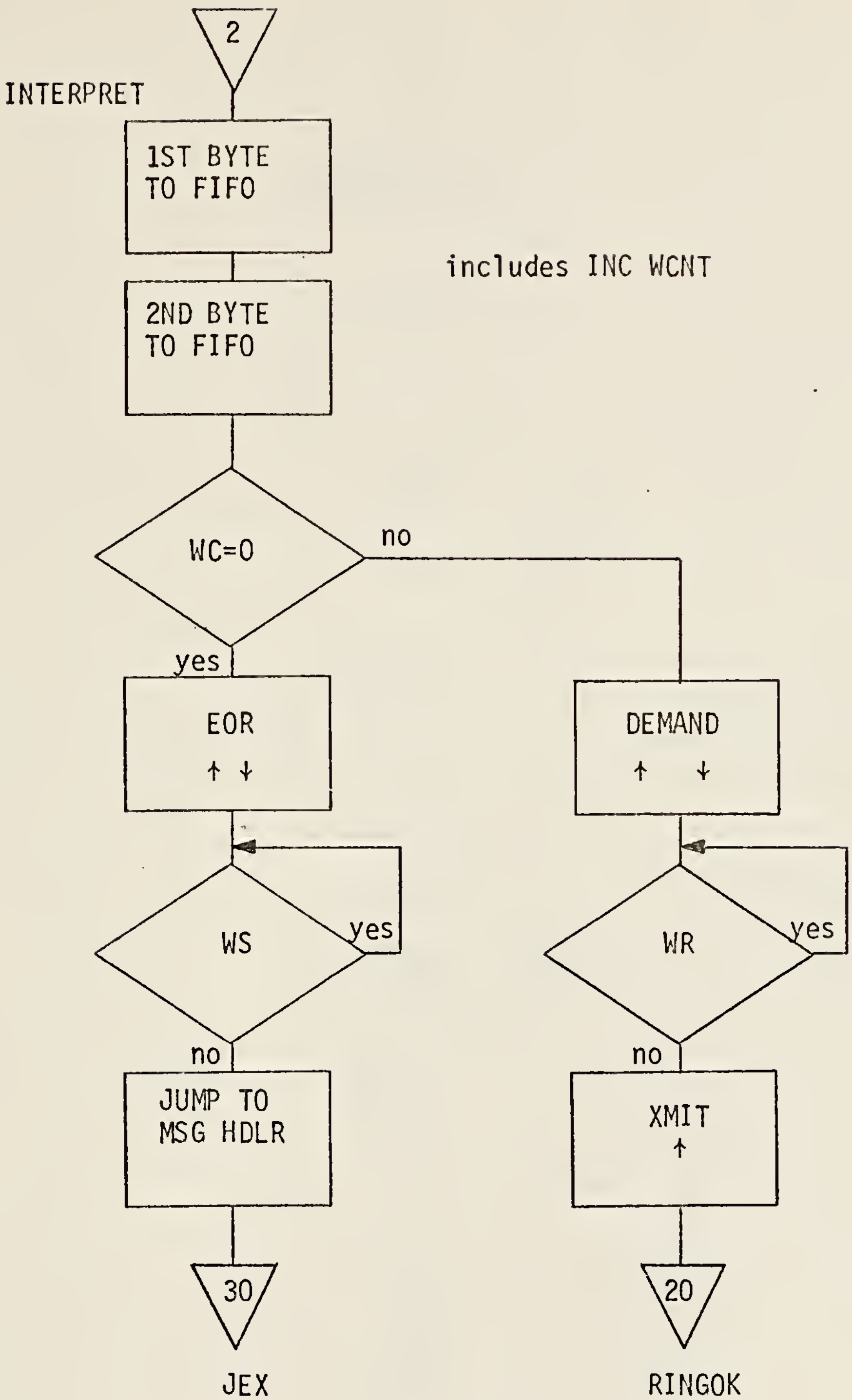


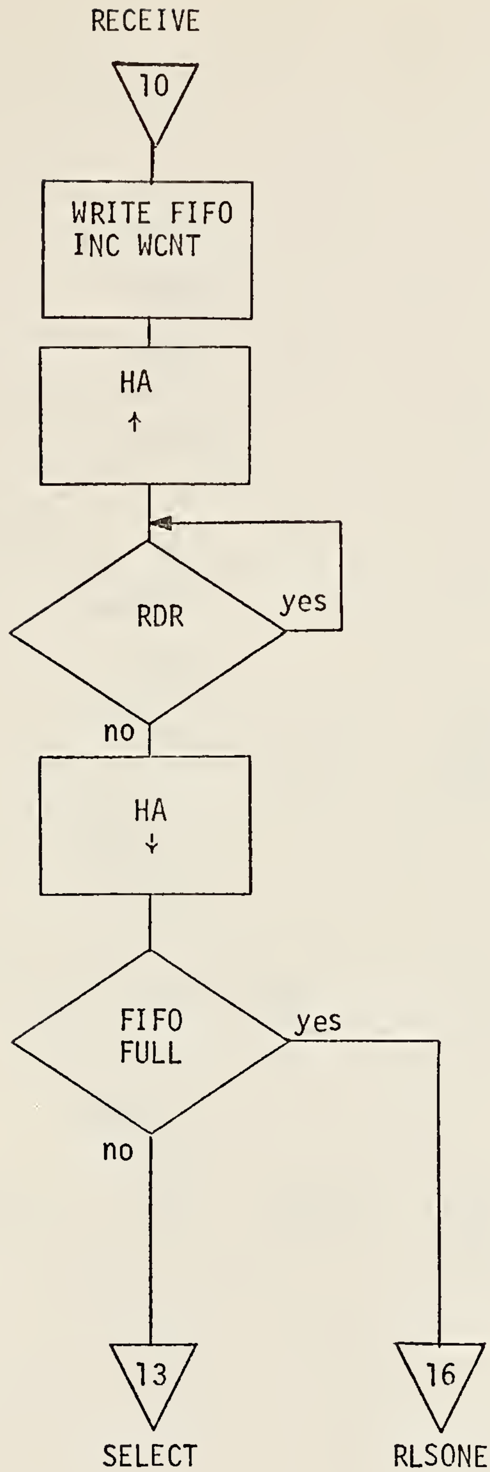
Fig. 19. RIHA Circuitry II

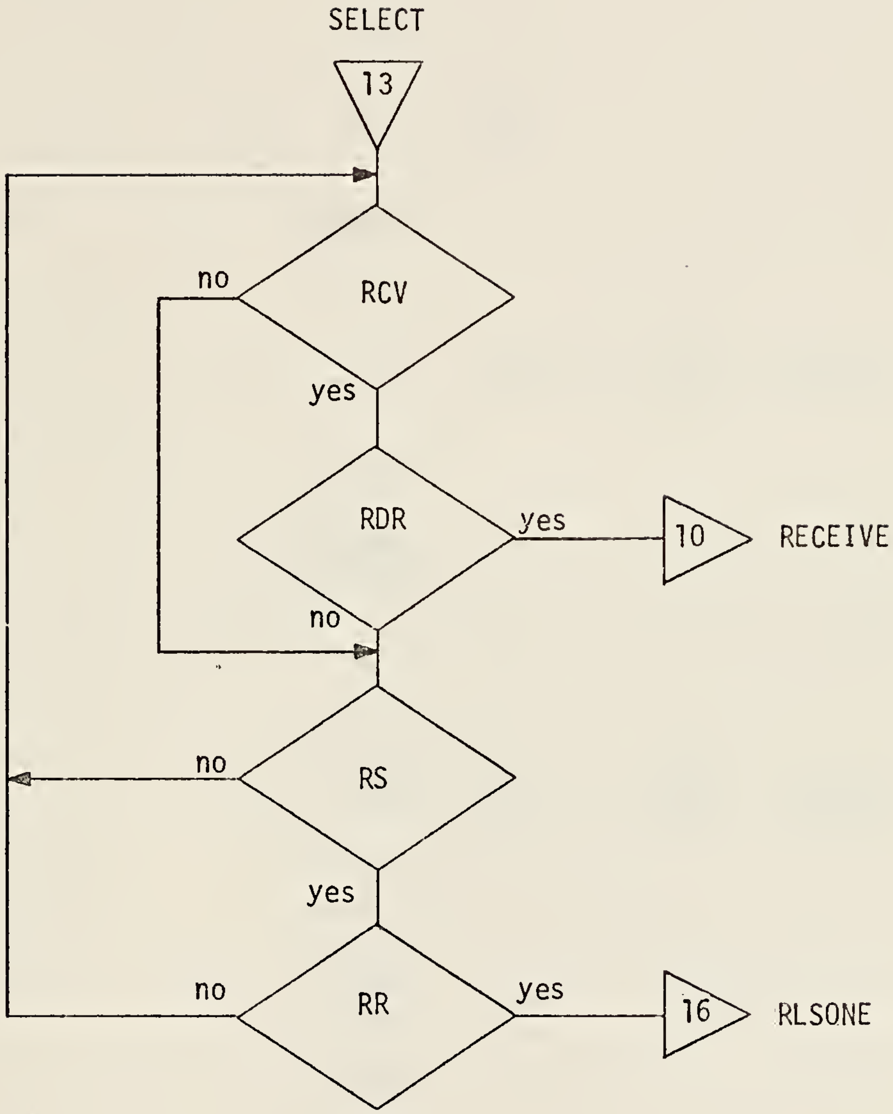
	Signal	Microcontroller Board	RIHA Board
Microcontroller "IN" Lines:	WS	5	17
	RS	6	19
	WR	7	21
	RR	8	23
	WC=0	9	25
	CNTUNEQAL	12	33
	STATUSOK	13	35
	RCV	31	69
	RDR	30	71
	RID	29	73
	S3	28	75
	S4	27	77
	R4	45	66
	R5	46	68
	R6	47	70
	R7	48	72
Microcontroller "OUT" Lines:	WCNT	51	18
	RCNT	52	20
	SETB	53	22
	SEL41	60	41
	SEL21	61	43
	ERASE	62	45
	FIFORW	63	47
	STROBE41	64	49
	DEMSET	70	57
	EORSET	71	59
	EOFSET	72	61
	INTRPTSET	73	63
	XMITSET	76	85
	HDRSET	77	87
	HASET	78	89
	APNSET	79	91
	WRTNSET	80	93
	DISCSET	81	95
	RSTSET	82	97
	D0	90	90
	D1	89	88
	+5Volt	99	99
	+5Volt	100	100
	GND	3	3
	GND	4	4

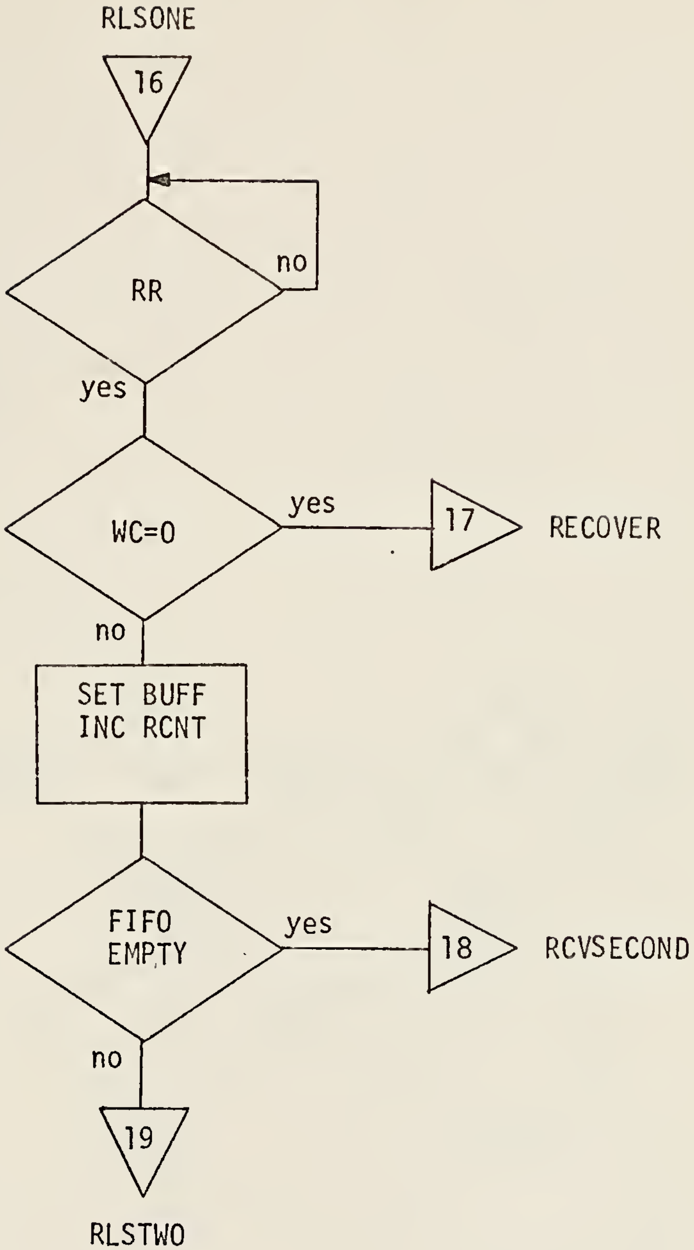
Figure 20. RIHA Pin Assignments

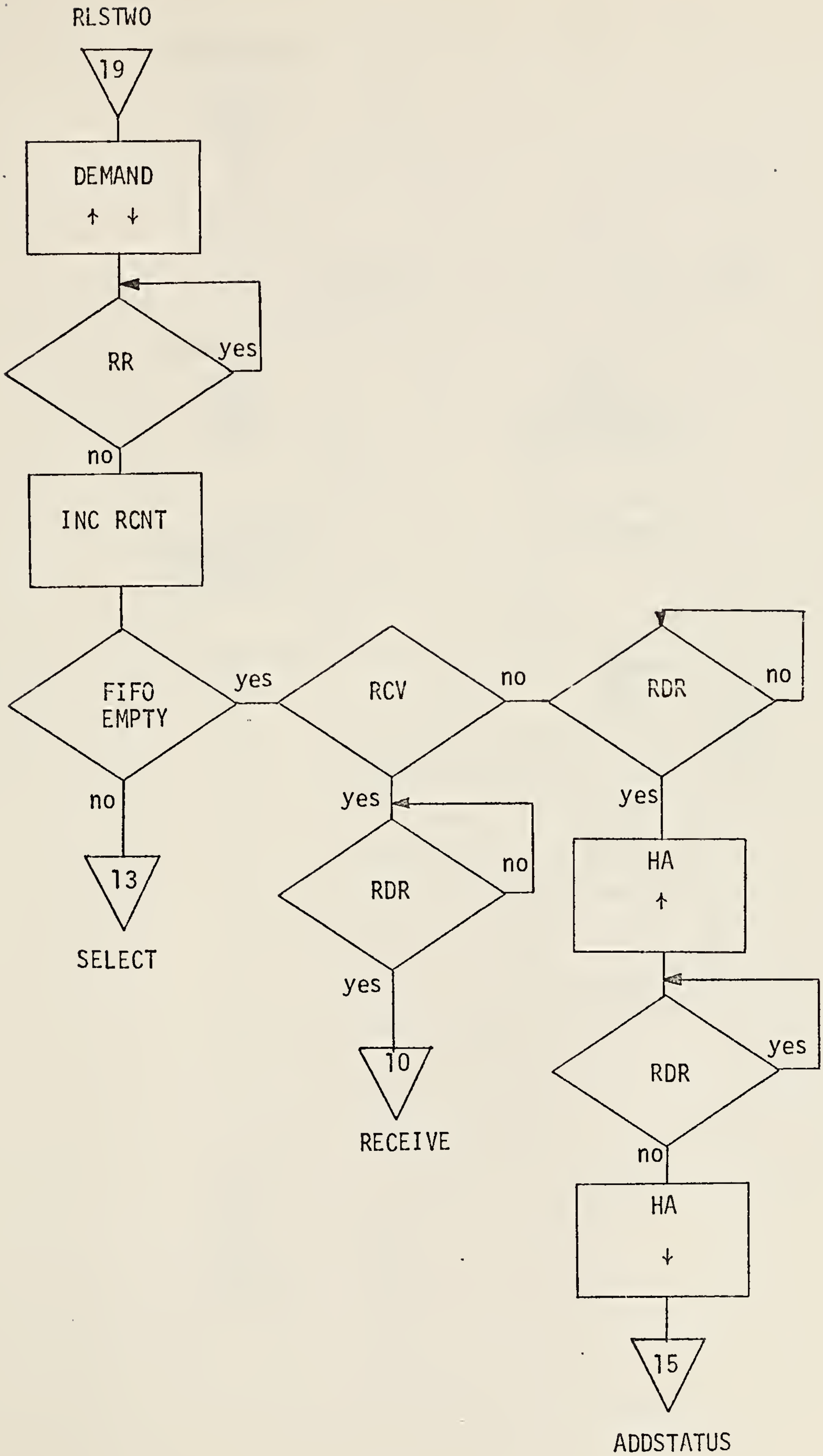

START
0
ERASE FIFO
WS
yes
no
RCV
no
yes
WR
no
yes
2
INTERPRET
RS
yes
no
INTERRUPT
↑ ↓
RDR
no
yes
10
RECEIVE

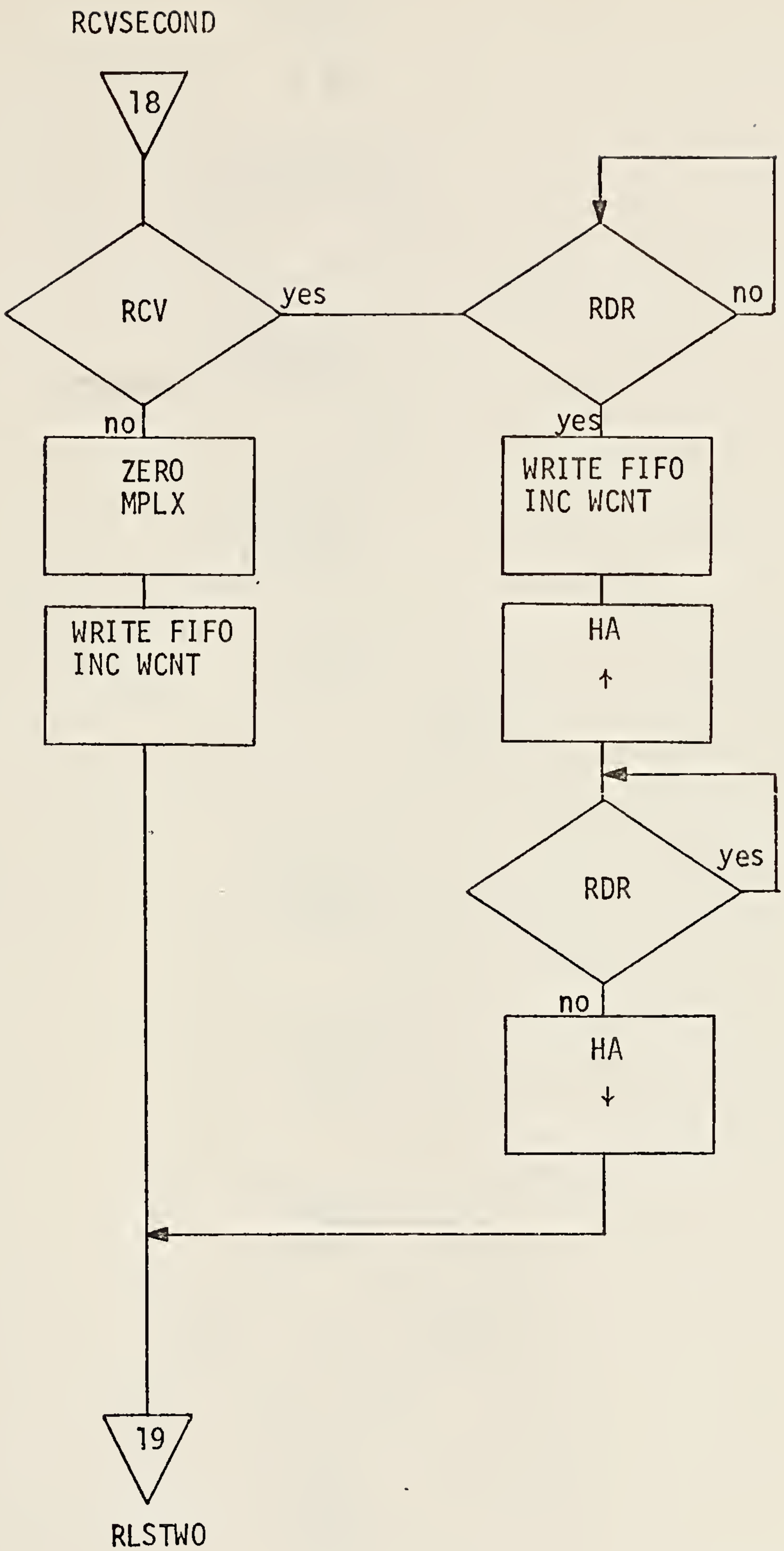

2
INTERPRET
1ST BYTE
TO FIFO
includes INC WCNT
2ND BYTE
TO FIFO
WC=0
no
yes
EOR
DEMAND
WS
yes
WR
yes
no
no
JUMP TO
MSG HDLR
XMIT
30
20
JEX
RINGOK

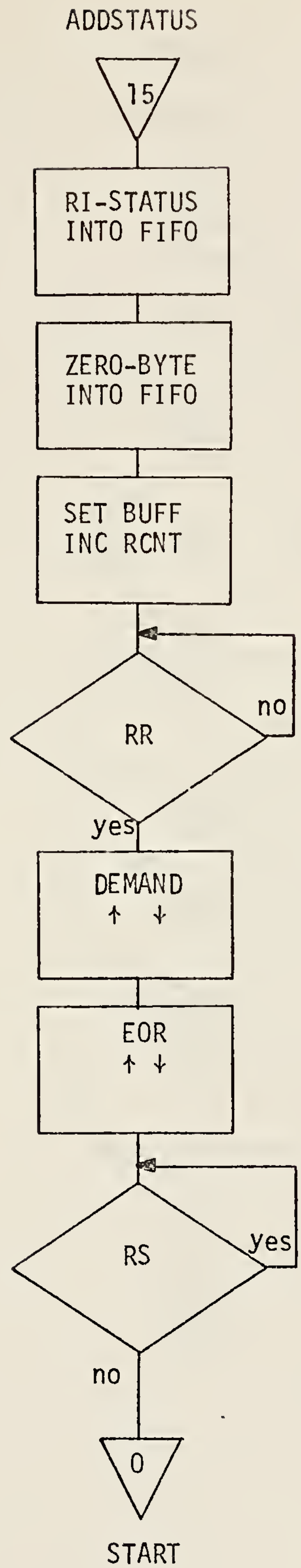

RECEIVE
10
WRITE FIFO
INC WCNT
HA
↑
RDR
yes
no
HA
↓
FIFO
FULL
yes
no
13
SELECT
16
RLSONE


SELECT
13
RCV
no
yes
RDR
yes
10
RECEIVE
no
RS
no
yes
RR
no
yes
16
RLSONE

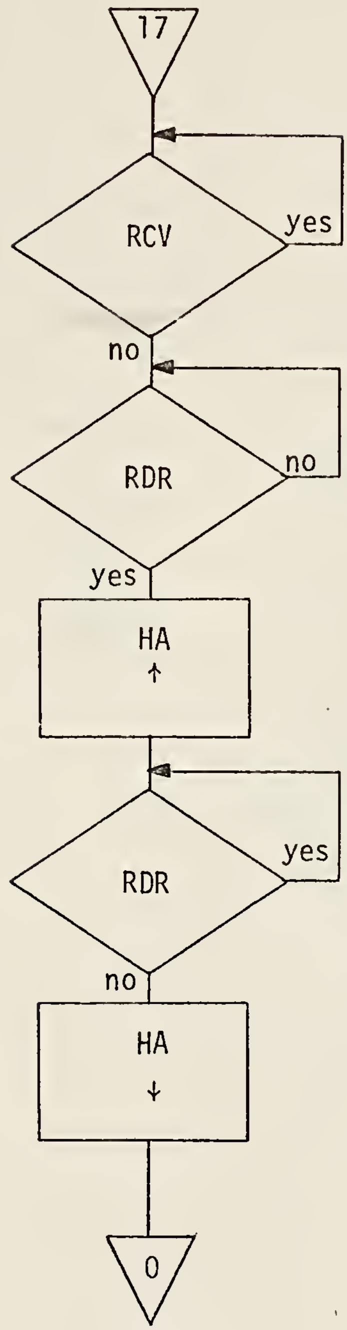

RLSONE
16
RR
no
yes
WC=0
yes
17
RECOVER
no
SET BUFF
INC RCNT
FIFO
EMPTY
yes
18
RCVSECOND
no
19
RLSTWO


RLSTWO
19
DEMAND
RR
yes
no
INC RCNT
FIFO EMPTY
yes
no
13
SELECT
RCV
no
yes
RDR
no
yes
10
RECEIVE
RDR
no
yes
HA
RDR
yes
no
HA
15
ADDSTATUS

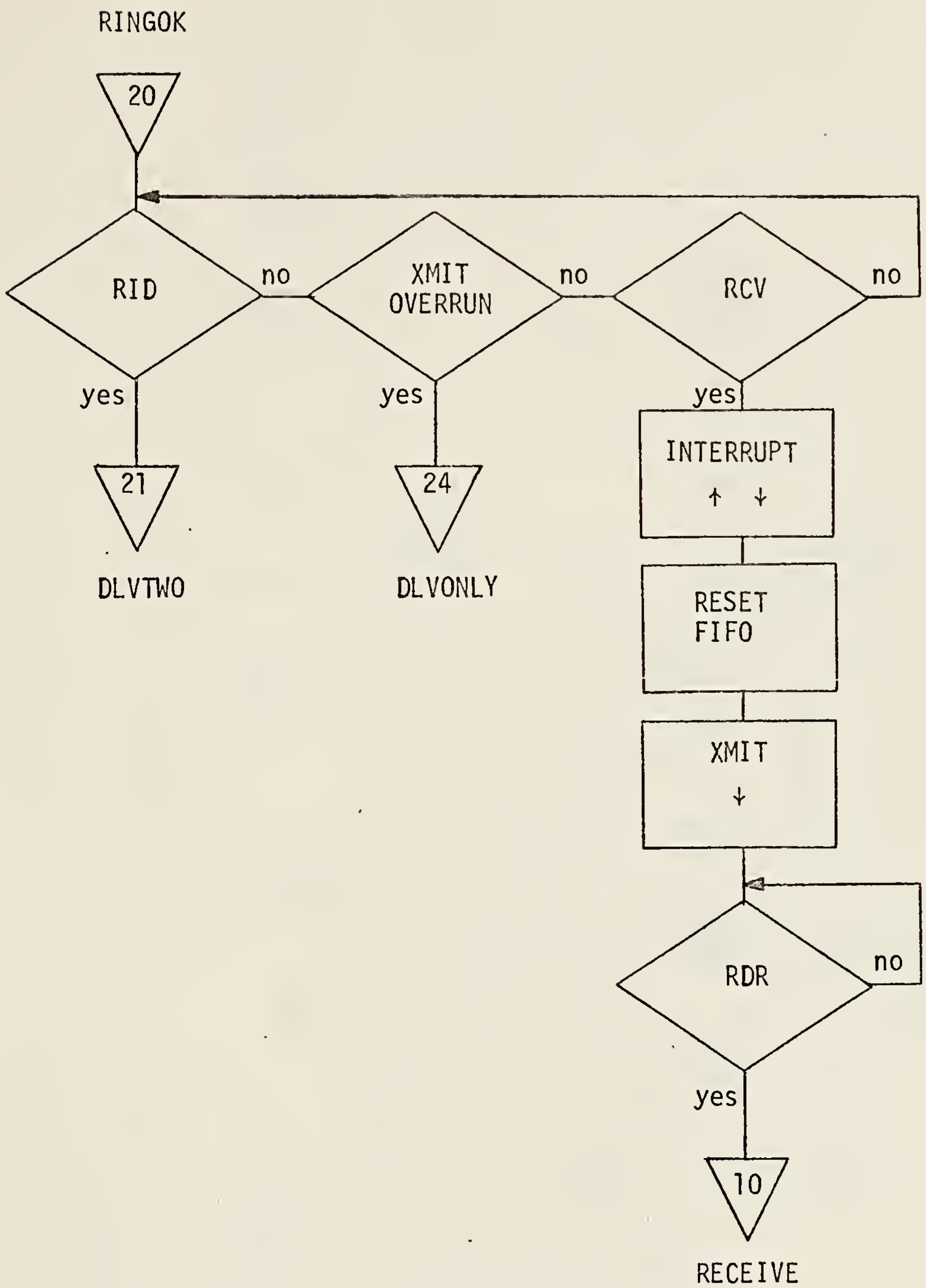

RCVSECOND
18
RCV
yes
no
ZERO MPLX
WRITE FIFO INC WCNT
RDR
no
yes
WRITE FIFO INC WCNT
HA ↑
RDR
yes
no
HA ↓
19
RLSTWO

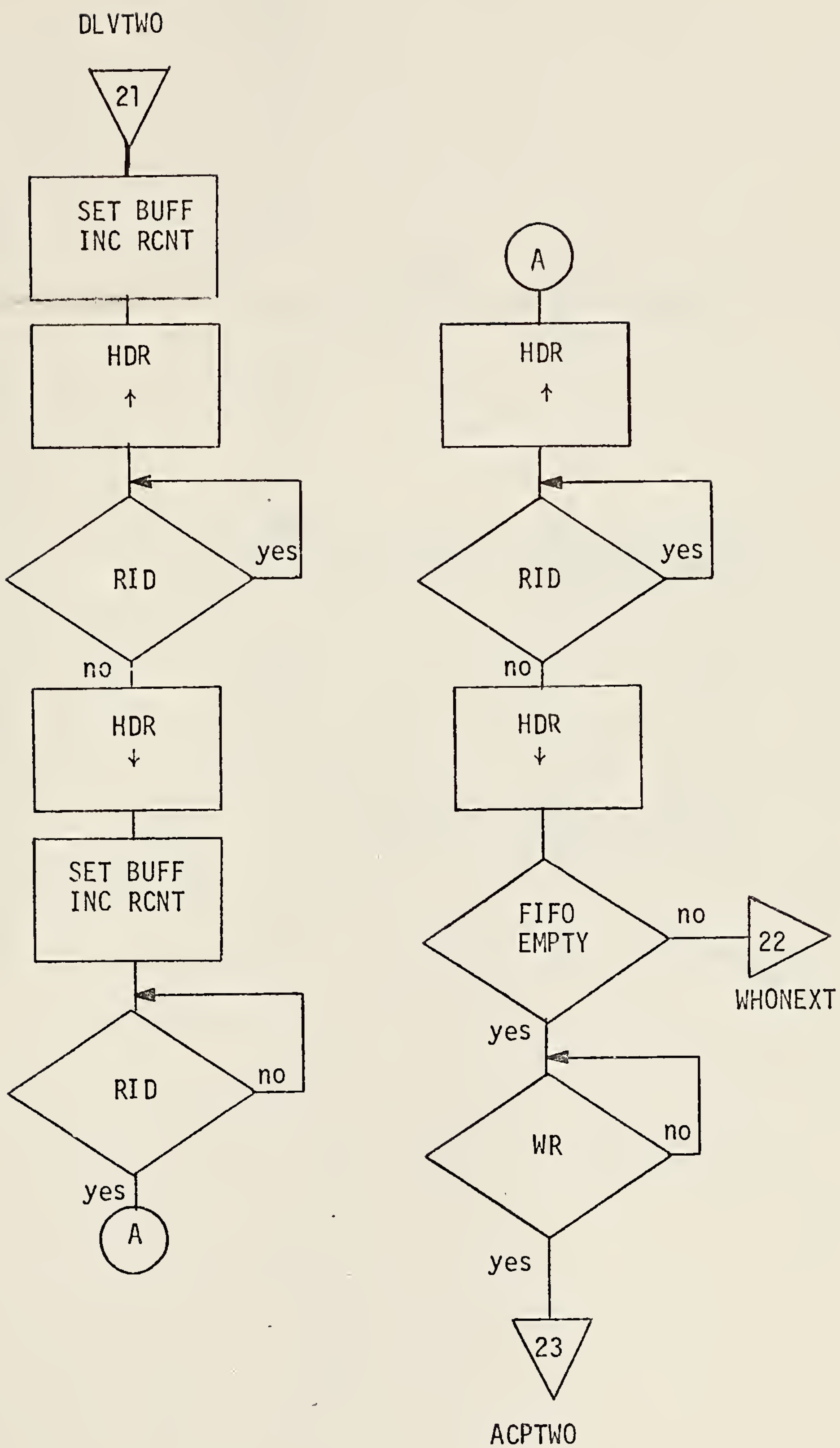

ADDSTATUS
15
RI-STATUS
INTO FIFO
ZERO-BYTE
INTO FIFO
SET BUFF
INC RCNT
RR
no
yes
DEMAND
↑ ↓
EOR
↑ ↓
RS
yes
no
0
START

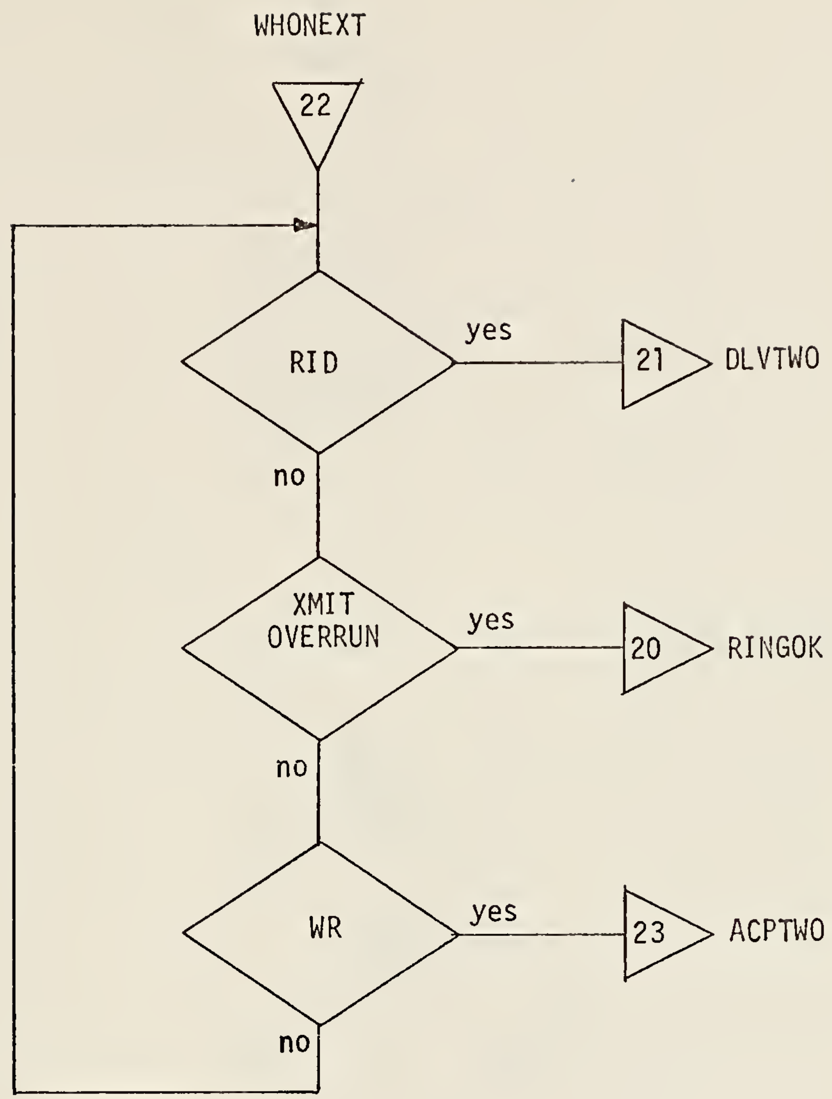
RECOVER

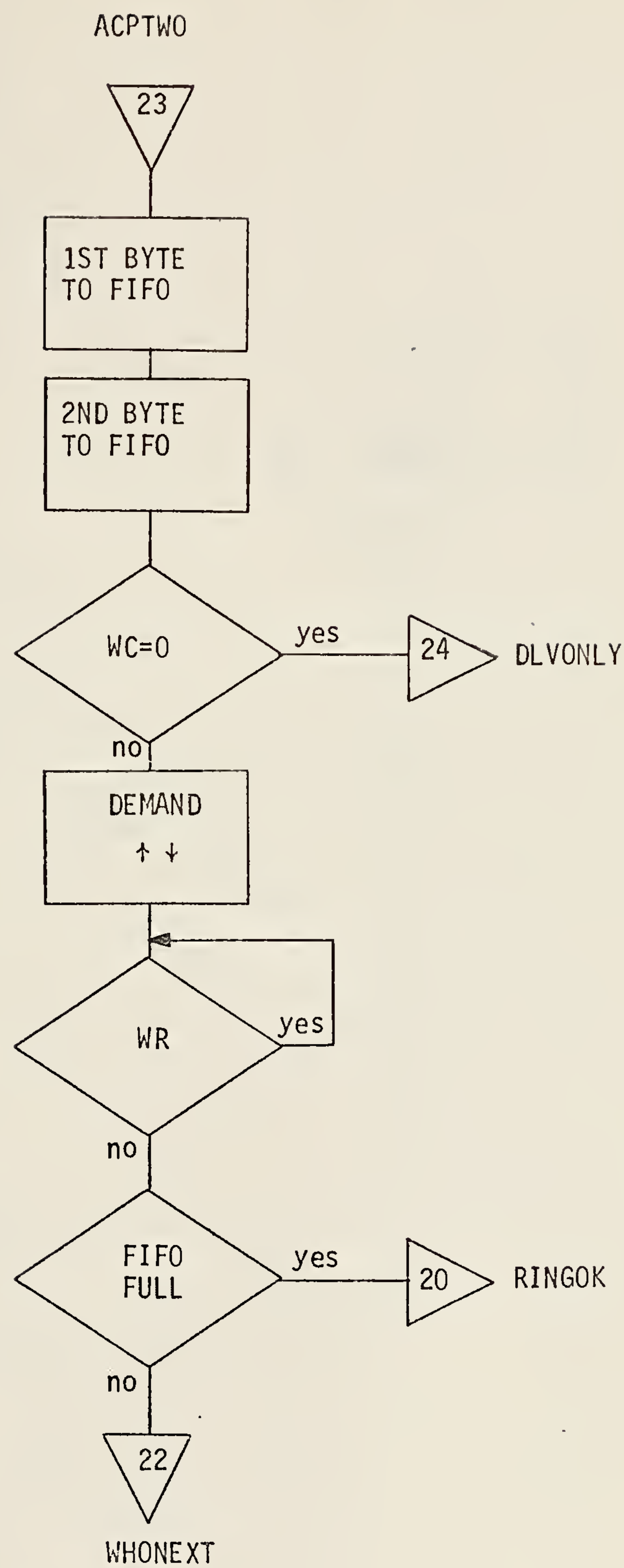


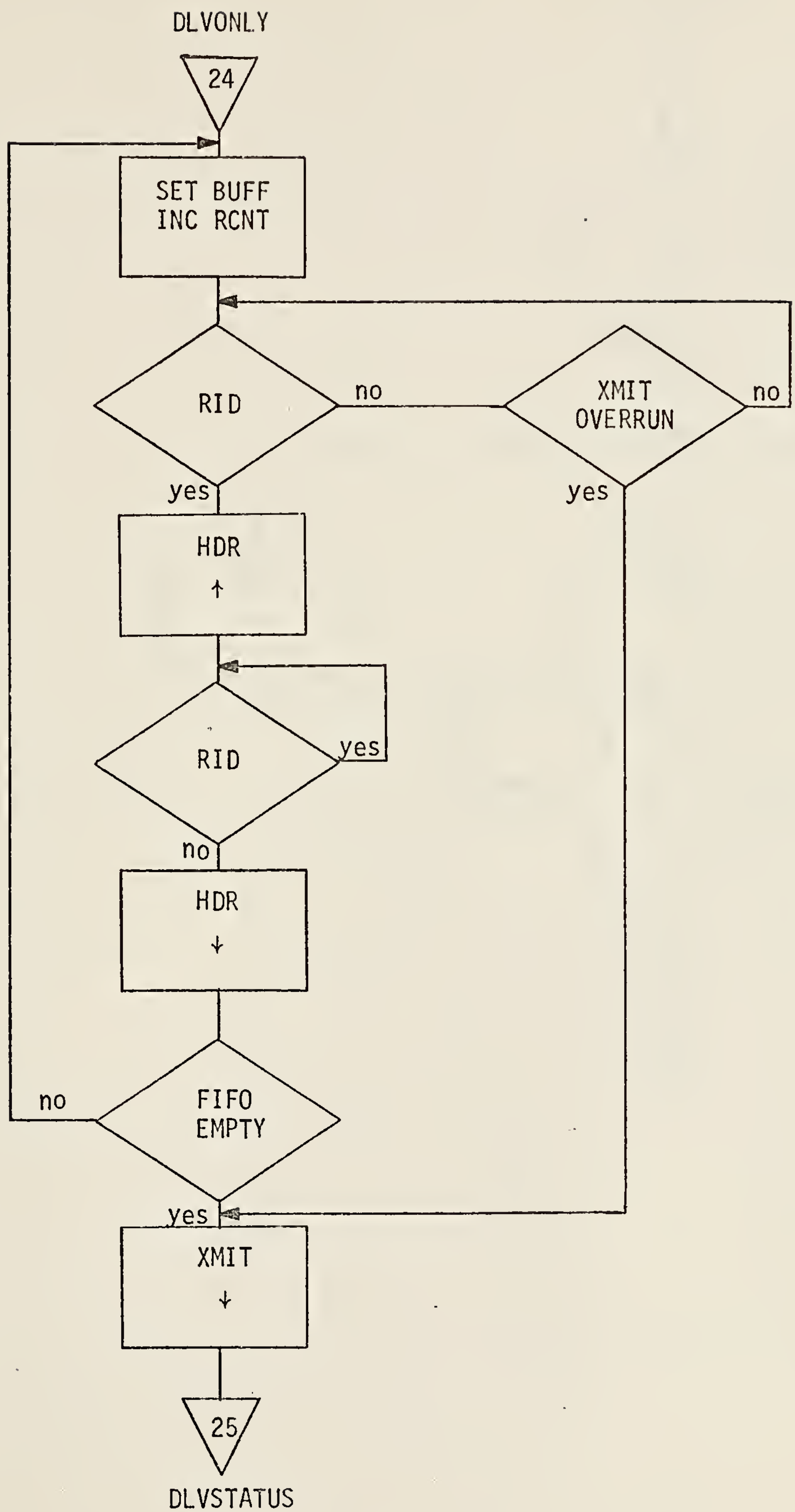
START

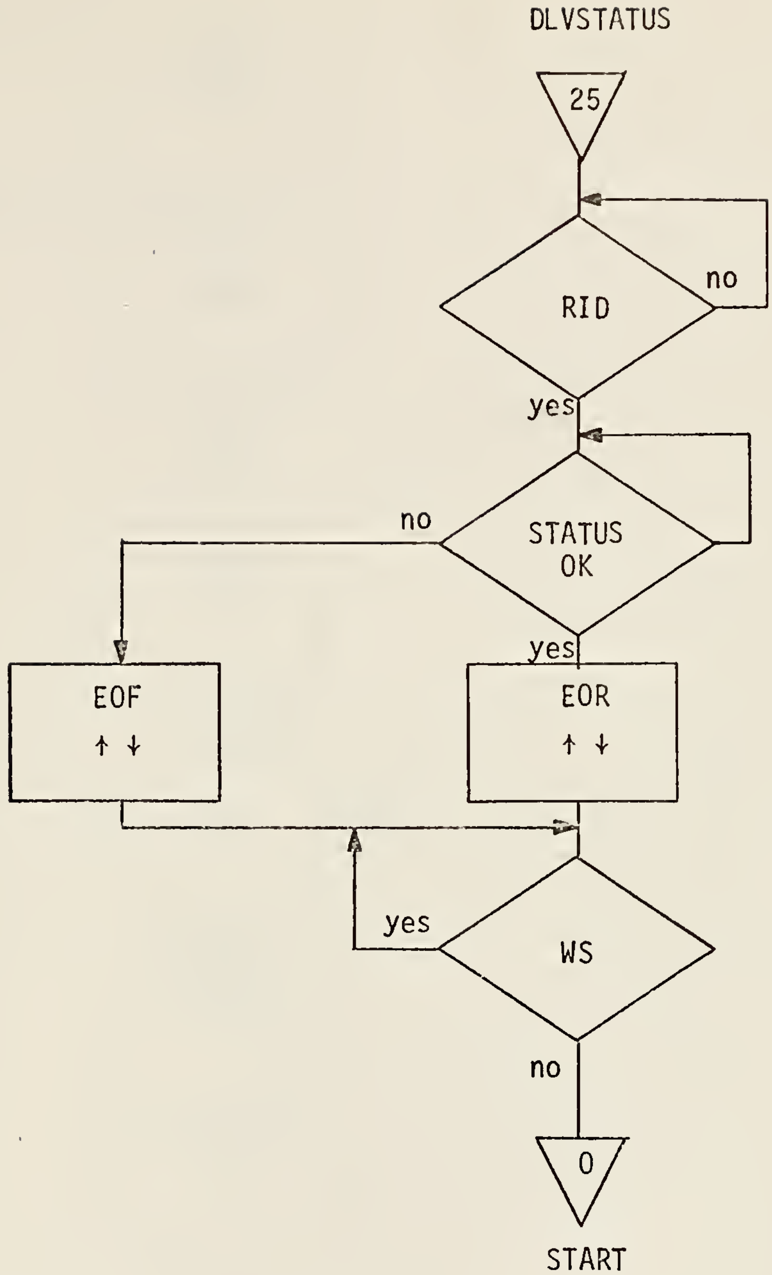

RINGOK
20
RID
no
XMIT
OVERRUN
no
RCV
no
yes
yes
yes
21
24
DLVTWO
DLVONLY
INTERRUPT
↑ ↓
RESET
FIFO
XMIT
↓
RDR
no
yes
10
RECEIVE

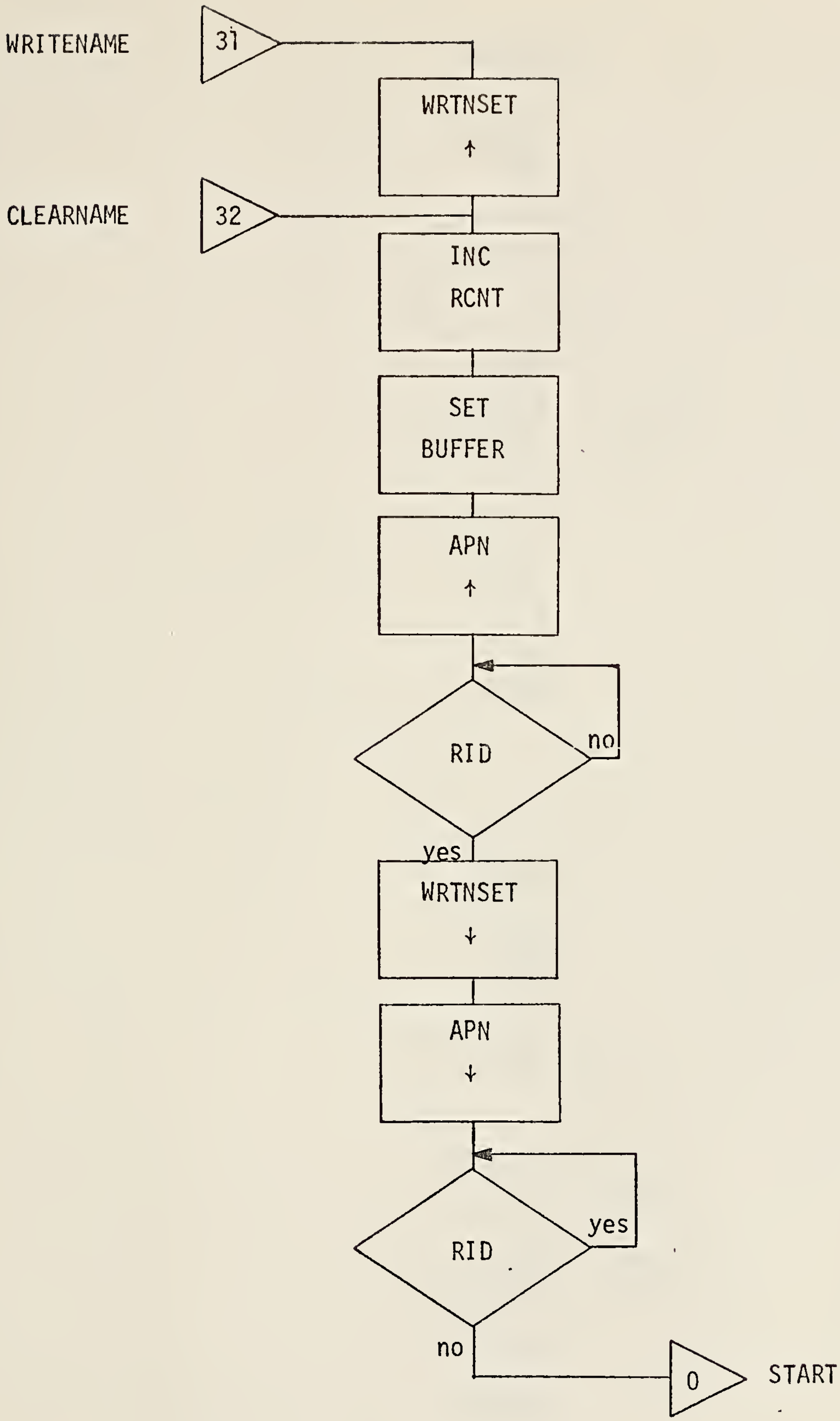

DLVTWO
21
SET BUFF
INC RCNT
HDR
↑
RID
yes
no
HDR
↓
SET BUFF
INC RCNT
RID
no
yes
A
A
HDR
↑
RID
yes
no
HDR
↓
FIFO
EMPTY
no
22
WHONEXT
yes
WR
no
yes
23
ACPTWO


WHONEXT
22
RID
yes
21
DLVTWO
no
XMIT
OVERRUN
yes
20
RINGOK
no
WR
yes
23
ACPTWO
no

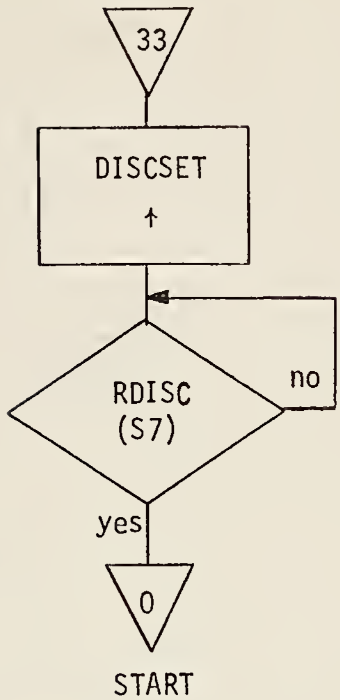

ACPTWO
23
1ST BYTE
TO FIFO
2ND BYTE
TO FIFO
WC=0
yes
24
DLVONLY
no
DEMAND
↑ ↓
WR
yes
no
FIFO
FULL
yes
20
RINGOK
no
22
WHONEXT


DLVONLY
24
SET BUFF
INC RCNT
RID
no
XMIT
OVERRUN
no
yes
yes
HDR
↑
RID
yes
no
HDR
↓
FIFO
EMPTY
no
yes
XMIT
↓
25
DLVSTATUS

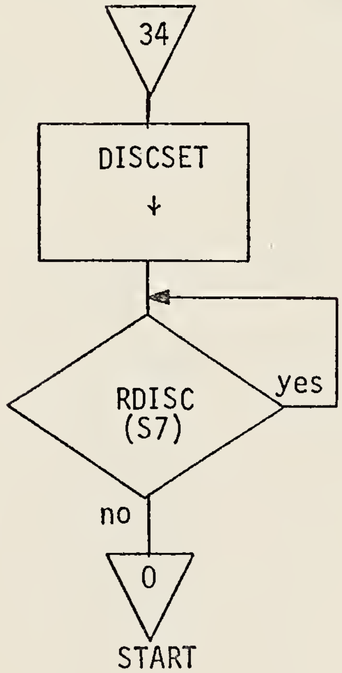

DLVSTATUS
25
RID
no
yes
STATUS OK
no
yes
EOF
↑ ↓
EOR
↑ ↓
WS
yes
no
0
START

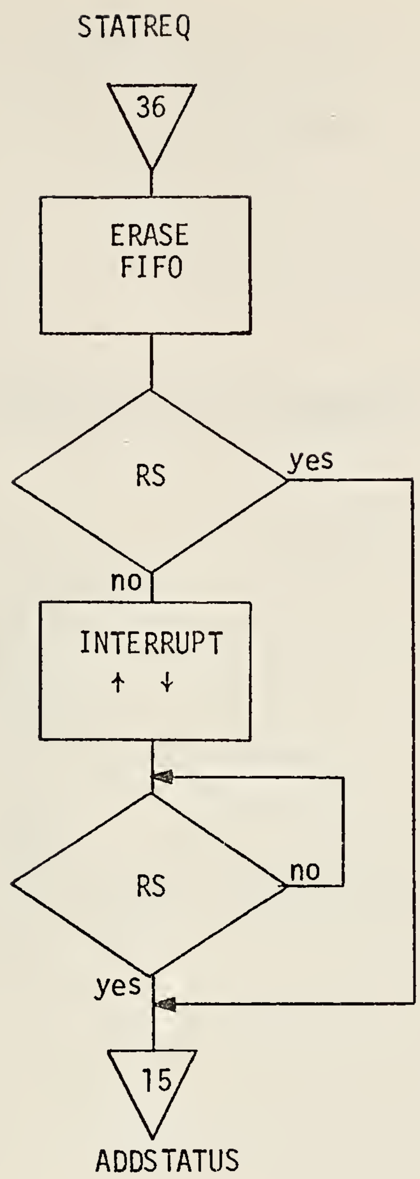

WRITENAME
31
WRTNSET
↑
CLEARNAME
32
INC
RCNT
SET
BUFFER
APN
↑
RID
no
yes
WRTNSET
↓
APN
↓
RID
yes
no
0
START

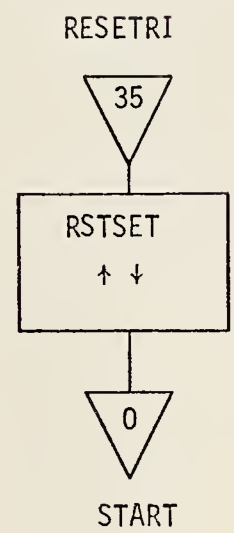
DISCONNECT



CONNECT




STATREQ
36
ERASE FIFO
RS
yes
no
INTERRUPT
↑ ↓
RS
no
yes
15
ADDSTATUS


RESETRI
35
RSTSET
↑ ↓
0
START

```
/ RIHA MICROCONTROLLER PROGRAM

/ OUTPUT BUS DATA ASSIGNMENTS

WRITE = 0
READ = 1
PULSE = 0
UP = 1
DOWN = 0
FIRSTBYTE = 0
SECONDBYTE = 1
RIDATA = 3
RISTATUS = 2
JEXTABLE = 0F0H
/ OUTPUT PORT NUMBER ASSIGNMENTS
WCNT = 0
RCNT = 1
SETB = 2
SEL41 = 9
SEL21 = 10
ERASE = 11
FIFORW = 12
STROBE41 = 13
DEMSET = 19
EORSET = 20
EOFSET = 21
INTRPTSET = 22
XMITSET = 25
HDRSET = 26
HASET = 27
APNSET = 28
WRTNSET = 29
DISCSET = 30
RSTSET = 31
/ INPUT PORT NUMBER ASSIGNMENTS
STATUSOK = 7
CNTUNEQAL = 8
WCZERO = 11
RR = 12
WR = 13
RS = 14
WS = 15
RCV = 21
RDR = 22
RID = 23
S3 = 24
S7 = 25

/ PROGRAM START

                        SEL41       := DOWN
                        SEL21       := DOWN
                        ERASE       := DOWN
                        FIFORW      := UP
                        STROBE41    := DOWN
                        DEMSET      := DOWN
                        EORSET      := DOWN
                        EOFSET      := DOWN
                        INTRPTSET   := DOWN
                        XMITSET     := DOWN
                        HDRSET      := DOWN
                        HASET       := DOWN
                        APNSET      := DOWN
                        WRTNSET     := DOWN
                        DISCSET     := DOWN
                        RSTSET      := DOWN
```

```
/ 0

START,       ERASE      := UP
             ERASE      := DOWN
IDLE,        WS         =: WAITXMIT
             -RCV       =: IDLE
             RS         =: WAITONRI
             INTRPTSET  := UP
             INTRPTSET  := DOWN
WAITONRI,    -RDR       =: *
                        =: RECEIVE
WAITXMIT,    -WR        =: *
                        =: INTERPRET

/ 2

INTERPRET,   SEL21      := WRITE
             SEL41      := FIRSTBYTE
             FIFORW     := WRITE
             FIFORW     := READ
             WCNT       := PULSE
             SEL41      := SECONDBYTE
             FIFORW     := WRITE
             FIFORW     := READ
             WCNT       := PULSE
             WCZERO     =: CONTROLMSG
             DEMSET     := UP
             DEMSET     := DOWN
             WR         =: *
             XMITSET    := UP
                        =: RINGOK
CONTROLMSG,  EORSET     := UP
             EORSET     := DOWN
             WS         =: *
             0          =::JEXTABLE

/ 10

RECEIVE,     SEL21      := WRITE
             SEL41      := RIDATA
             FIFORW     := WRITE
             FIFORW     := READ
             WCNT       := PULSE
             HASET      := UP
             RDR        =: *
             HASET      := DOWN
             CNTUNEQAL  =: SELECT
                        =: RLSONE

/ 13

SELECT,      -RCV       =: SELECTRLS
             RDR        =: RECEIVE
SELECTRLS,   -RS        =: SELECT
             RR         =: RLSONE
                        =: SELECT

/ 16

RLSONE,      -RR        =: *
             WCZERO     =: RECOVER
             SEL21      := READ
             SETB       := PULSE
             RCNT       := PULSE
             CNTUNEQAL  =: RLSTWO
                        =: RCVSECOND
```

```
/ 19

RLSTWO,         SEL21        := READ
                DEMSET       := UP
                DEMSET       := DOWN
                RR           =: *
                RCNT         := PULSE
                CNTUNEQAL    =: SELECT
                -RCV         =: RCVEND
                -RDR         =: *
                             =: RECEIVE
RCVEND,         -RDR         =: *
                HASET        := UP
                RDR          =: *
                HASET        := DOWN
                             =: ADDSTATUS


/ 18

RCVSECOND,      -RCV         =: ZEROBYTE
                -RDR         =: *
                SEL21        := WRITE
                SEL41        := RIDATA
                FIFORW       := WRITE
                FIFORW       := READ
                WCNT         := PULSE
                HASET        := UP
                RDR          =: *
                HASET        := DOWN
                             =: RLSTWO
ZEROBYTE,       SEL21        := WRITE
                STROBE41     := UP
                FIFORW       := WRITE
                FIFORW       := READ
                WCNT         := PULSE
                STROBE41     := DOWN
                             =: RLSTWO

/ 15

ADDSTATUS,      SEL21        := WRITE
                SEL41        := RISTATUS
                FIFORW       := WRITE
                FIFORW       := READ
                WCNT         := PULSE
                STROBE41     := UP
                FIFORW       := WRITE
                FIFORW       := READ
                STROBE41     := DOWN
                WCNT         := PULSE
                SEL21        := READ
                SETB         := PULSE
                RCNT         := PULSE
                -RR          =: *
                DEMSET       := UP
                DEMSET       := DOWN
                EORSET       := UP
                RS           =: *
                EORSET       := DOWN
                RCNT         := PULSE
                             =: START
```

```
/ 17

RECOVER,        RCV          =: *
                -RDR         =: *
                HASET        := UP
                RDR          =: *
                HASET        := DOWN
                             =: START

/ 20

RINGOK,         RID          =: DLVTWO
                S3           =: DLVONLY
                -RCV         =: RINGOK
                INTRPTSET    := UP
                INTRPTSET    := DOWN
                ERASE        := UP
                ERASE        := DOWN
                XMITSET      := DOWN
                -RDR         =: *
                             =: RECEIVE

/ 21

DLVTWO,         SEL21        := READ
                SETB         := PULSE
                RCNT         := PULSE
                HDRSET       := UP
                RID          =: *
                HDRSET       := DOWN
                SETB         := PULSE
                RCNT         := PULSE
                -RID         =: *
                HDRSET       := UP
                RID          =: *
                HDRSET       := DOWN
                CNTUNEQAL    =: WHONEXT
                -WR          =: *
                             =: ACPTWO

/ 22

WHONEXT,        RID          =: DLVTWO
                S3           =: RINGOK
                WR           =: ACPTWO
                             =: WHONEXT

/ 23

ACPTWO,         SEL21        := WRITE
                SEL41        := FIRSTBYTE
                FIFORW       := WRITE
                FIFORW       := READ
                WCNT         := PULSE
                SEL41        := SECONDBYTE
                FIFORW       := WRITE
                FIFORW       := READ
                WCNT         := PULSE
                WCZERO       =: DLVONLY
                DEMSET       := UP
                DEMSET       := DOWN
                WR           =: *
                CNTUNEQAL    =: WHONEXT
                             =: RINGOK
```

```
/ 24

DLVONLY,        SEL21        := READ
                SETB         := PULSE
TESTRID,        RID          =: NOXOVR
                S3           =: XMITEND
                             =: TESTRID
NOXOVR,         HDRSET       := UP
                RID          =: *
                HDRSET       := DOWN
                RCNT         := PULSE
                CNTUNEQAL    =: DLVONLY
XMITEND,        XMITSET      := DOWN
                             =: DLVSTATUS

/ 25

DLVSTATUS,      -RID         =: *
                -STATUSOK    =: EXCEPTION
                EORSET       := UP
                EORSET       := DOWN
WSTEST,         WS           =: *
                             =: START
EXCEPTION,      EOFSET       := UP
                EOFSET       := DOWN
                             =: WSTEST

/ 31

WRITENAME,      WRTNSET      := UP

/ 32

CLEARNAME,      RCNT         := PULSE
                SEL21        := READ
                SETB         := PULSE
                APNSET       := UP
                -RID         =: *
                WRTNSET      := DOWN
                APNSET       := DOWN
                RID          =: *
                             =: START

/ 33

DISCONNECT,     DISCSET      := UP
                -S7          =: *
                             =: START

/ 34

CONNECT,        DISCSET      := DOWN
                S7           =: *
                             =: START

/ 36

STATREQ,        ERASE        := UP
                ERASE        := DOWN
                RS           =: ADDSTATUS
                INTRPTSET    := UP
                INTRPTSET    := DOWN
                -RS          =: *
                             =: ADDSTATUS
```

```
/ 35
RESETRI,        RSTSET     := UP
                RSTSET     := DOWN
                           =: START

/ 30
JEXTABLE,                  =: WRITENAME
                           =: CLEARNAME
                           =: DISCONNECT
                           =: CONNECT
                           =: STATREQ
                           =: RESETRI

$$
```

BIBLIOGRAPHY

1. Brubaker Jr., R.H., A General Purpose Microcontroller, paper prepared at Computer Science Group, Naval Postgraduate School, Monterey, California, March 1974.

2. Kildall, G.A., A Symbolic Microcontroller Assembly Language, Computer Science Group (Internal Document), Naval Postgraduate School, Monterey, California, April 1974.

3. Hirt, K.A., A Prototype Ring-Structured Computer Network Using Micro-Computers, Masters Thesis, Naval Postgraduate School, Monterey, California 1973.

4. Intel Corporation, A Guide to PL/M Programming, September 1973.

5. Intel Corporation, MCS - 8 Microcomputer Set, User's Manual, November 1973.

6. Harris, M.S., A Prototype Ring Interface for the NPS Data Communication Ring, Masters Thesis, Naval Postgraduate School, Monterey, 1974.

7. IBM System /360 I/O Interface, Channel to Control Unit, OEM Information, A22-6843-3.

8. IBM 2701 Data Adapter Unit, OEM Information, GA22-6844.

9. IBM 2701 Data Adapter Unit, Component Description, GA22-6864.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0212 Naval Postgraduate School Monterey, California 93940	2
3.	Chairman, Computer Science Group, Code 72 Computer Science Group Naval Postgraduate School Monterey, California 93940	1
4.	Asst. Prof. R. H. Brubaker,Jr. Code 72Bh Computer Science Group Naval Postgraduate School Monterey, California 93940	1
5.	Prof. G. Barksdale, Code 72 Computer Science Group Naval Postgraduate School Monterey, California 93940	1
6.	Marineamt-Al- 294 Wilhelmshaven Federal Republic of Germany	1
7.	Dokumentationszentrale der Bundeswehr(See) 53 Bonn Friedrich-Ebert-Allee 34 Federal Republic of Germany	1
8.	LCDR Eberhard O. Wortmann, FGN 239 Flensburg Beethovenstrasse 17 Federal Republic of Germany	1

153888

Thesis
W8754 Wortmann
c.1
Design and implementation of a ring interface/host adapter for an IBM system 360.